

# **AppleWriter™ Cookbook**

**Don Lancaster**





# **AppleWriter Cookbook**





# AppleWriter<sup>TM</sup> Cookbook

---

*Don Lancaster*

**Howard W. Sams & Co., Inc.**

A Subsidiary of Macmillan, Inc.

4300 West 62nd Street, Indianapolis, Indiana 46268 U.S.A.

## Trademarks

All terms mentioned in this book that are known to be trademarks or service marks are listed below. In addition, terms suspected of being trademarks or service marks have been appropriately capitalized. Howard W. Sams & Co., Inc., cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Apple, Applesoft, ProDOS, and ImageWriter are registered trademarks and Appleworks, AppleWriter, and Laserwriter are trademarks of Apple Computer, Inc.

Copy II Plus is a trademark of Central Point Software Inc.

Diablo is a trademark of XEROX Corporation.

Epson is a registered trademark of Epson America, Inc.

Grappler is a registered trademark of Orange Micro Inc.

Hayes is a registered trademark of Hayes Microcomputer Products Inc.

NEC is a registered trademark of NEC Information Systems.

Qume and Sprint are registered trademarks of Qume Corporation.

Radio Shack is a registered trademark of the Tandy Corporation.

Spinwriter is a registered trademark of Nippon Electric Company.

Visicalc is a registered trademark of VisiCorp, Inc.

Zork is a registered trademark of Infocom Inc.

*The opinions and editorial conventions embodied in the text reflect the preferences of the author. No endorsement by Howard W. Sams & Co. is implied.*

Copyright © 1986 by Don Lancaster

FIRST EDITION

FIRST PRINTING—1986

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, neither the author nor the publisher assumes any responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-22460-7

Library of Congress Catalog Card Number: 85-72103

Printed in the United States of America

# Contents

Introduction . . . . .	.vi
------------------------	-----

1 Applewriter Answers from the Gila Helpline . . . . .	1
--	---

How Can I See Exactly What Will Be Printed?, 3 □ Tell Me More About .pd8, 4 □ How Do I Print One Middle Page of a Long File?, 4 □ How Do I Imbed NULLs?, 4 □ How Do I Produce Underlining and Superscripting on an Epson?, 5 □ How Do I Produce Superscripting on an Apple DMP?, 6 □ How Do I Solve the Shortline Problem?, 6 □ What About Multicharacter Imbedded Escape Commands?, 6 □ How Do I Improve Underlining?, 7 □ How Do I Make Backup Copies?, 7 □ What Is the Grappler Problem?, 7 □ Can I Link an Assembler to Applewriter?, 7 □ Can I Process Pictures Rather Than Words?, 8 □ How Can I Make My Own Patches?, 8 □ Show Me a PATCHIFIER Example, 8 □ What Trashes the IIc Status Display?, 9 □ How Can I Print a Line of Dots?, 9 □ How Do I Make 64 Identical Labels?, 9 □ What Copy Options Do I Have?, 10 □ What Trashes My 'Text Files?, 10 □ What if the Text File Really Is Sick?, 11 □ How Can I Put a Catalog in My 'Text File?, 11 □ How Can I Improve the Catalog Display?, 11 □ Can I Run a WPL Program While in Mid-Print?, 12 □ How Can I Search or Replace Backspace Commands?, 12 □ Why Can't I Print {<?, 12 □ How Can I Improve Daisywheel Underlining?, 13 □ How Can I Improve My Daisywheel Print Quality?, 13 □ How Do I Do Kerning?, 13 □ How Can I Put Comments in the Glossary?, 13 □ Why Do Some Glossary Strings Execute Immediately?, 14 □ How Do I Imbed a [V] in a File?, 14 □ How Can I Fake Superscripting or Subscripting?, 14 □ How Do I Print Several Columns?, 15 □ How Do I Improve Daisywheel Registration?, 15 □ How Do I Load a WPL String from Text?, 15

## 2 More Applewriter Answers from the Gila Helpline . . . . . 17

How Do You Make a Glossary Self Prompting?, 19 ☐ How Do I Solve the ProDOS 2.0 Prefix Hassle?, 20 ☐ Why Do Often Used Glossaries Get Longer?, 20 ☐ How Can I Print All of My Daisywheel Spokes?, 20 ☐ Why Does the BOLD PS Printwheel Foul Up Punctuation?, 21 ☐ What Causes the Diablo 630 Secondline Problem?, 22 ☐ What Is Meant by the Words Diablo Compatible?, 22 ☐ How Can I Proportionally Space and Microjustify?, 22 ☐ What Difference Does Extended Memory Make?, 23 ☐ What Programs Must I Keep on the Boot Disk?, 23 ☐ What Is Special About Applewriter IIe DOS?, 23 ☐ How Can I View the Program?, 24 ☐ How Do I Convert My Binary Files to Text Files?, 24 ☐ What Makes the [Closed Apple] Key "Stick"?, 24 ☐ How Do I Convert My Applewriter III Files?, 25 ☐ How Do I Read or Edit a Very Long Text File?, 25 ☐ What's the Fastest Way to Clear to End of Document?, 25 ☐ How Can I Imbed Hidden Lines in a Mailing List?, 26 ☐ So How Do I Print the Hidden Lines on a Mailing List?, 26 ☐ How Do I Add a Common Header to a Mailing List?, 26 ☐ How Do I Get the Page Numbers to Print?, 27 ☐ How Can I Print Double Headers or Footers?, 27 ☐ How Can I Print a Glossary or a WPL File?, 27 ☐ How Do I Do a HIRES Dump?, 28 ☐ How Do I Center a Title?, 28 ☐ Why Won't My IIC Print Past Line 80?, 28 ☐ Is WPL Really That Great?, 29 ☐ What Good Is Right Justification?, 29 ☐ How Do I Run Old Applewriter 2.0 on a IIe?, 29 ☐ How Do I Use Old Applewriter 2.0 to . . ., 29 ☐ How Do I Ring the Ding-Dong from WPL?, 30 ☐ What Causes Page Creep?, 30 ☐ How Can I Stop a Hyper [delete] Key?, 31 ☐ How Do I Imbed a Carriage Return in the Glossary?, 31 ☐ What Good Are the Apple Keys?, 31 ☐ Can I Preboot ProDOS Applewriter 2.0?, 32 ☐ Is Source Code Available?, 32 ☐ What Causes Missed Characters?, 32 ☐ What Causes Painfully Slow AWIIe Entry?, 33 ☐ Why Can't I Do a Decent Underline?, 33 ☐ How Can I Print a Solid Bullet?, 34 ☐ Is There an Easy Way to Do Form Letters?, 34 ☐ How Do I Imbed Escape Sequences?, 35 ☐ Why Do Some AWIIe Patches Disable the Help Screen?, 35 ☐ Why Does [Y] Sometimes Destroy Everything?, 35

## 3 Secrets of Top Quality Printing . . . . . 37

Four Print Quality Rules, 39 ☐ For Still More Print Quality, 48 ☐ That Old WD40 Ploy, 52 ☐ Imbedding Print Commands, 53 ☐ Verbatim Method, 56 ☐ Imbedding with the Glossary, 60 ☐ A Glossary Example, 61 ☐ Imbedding with WPL, 64 ☐ Camera-Ready Print Quality, 64

<b>4</b>	<b>Microjustification and Proportional Spacing. . . . .</b>	<b>67</b>
	WPL Custom Formatting, 69	
<b>5</b>	<b>Self Prompting a Glossary . . . . .</b>	<b>79</b>
	What Is in a Glossary?, 81 <input type="checkbox"/> Immediate WPL Execution from the Glossary, 82 <input type="checkbox"/> Glossary Restrictions, 84 <input type="checkbox"/> Self Titling and Self Prompting, 86 <input type="checkbox"/> Four Examples, 88	
<b>6</b>	<b>Some Patches . . . . .</b>	<b>91</b>
	AWIIe Patches, 93 <input type="checkbox"/> ProDOS 2.0 Patches, 102 <input type="checkbox"/> Wrap-Up, 108	
<b>7</b>	<b>Tearing into ProDOS Applewriter Version 2.0. . . . .</b>	<b>109</b>
	Analyzing ProDOS Applewriter 2.0, 113 <input type="checkbox"/> ProDOS MLI Links, 136 <input type="checkbox"/> Monitor Access, 138 <input type="checkbox"/> Memory Management, 140 <input type="checkbox"/> Character Entry, 141 <input type="checkbox"/> Screen Display, 143 <input type="checkbox"/> Individual Control Commands, 145 <input type="checkbox"/> Printing, 149 <input type="checkbox"/> WPL, 152	
<b>8</b>	<b>Capturing ProDOS Applewriter Version 2.0 Source Code . . . . .</b>	<b>159</b>
	Customizing ProDOS Applewriter 2.0, 161 <input type="checkbox"/> Why Modify?, 162 <input type="checkbox"/> A Final Plum, 167 <input type="checkbox"/> A Wish List, 168	
<b>Appendixes</b>		
<b>A</b>	<b>WPL Programs and Applesoft Patches</b>	<b>171</b>
<b>B</b>	<b>Machine Language Patches . . . . .</b>	<b>201</b>
<b>C</b>	<b>Internal ProDOS Applewriter 2.0 Program Details. . . . .</b>	<b>223</b>
	<b>Index . . . . .</b>	<b>329</b>



# Introduction

I have been assisting the Gila Valley Apple Growers Association people in maintaining an Applewriter® voice helpline at (602) 428-4073. After working with this free service for a while, I found out what people really want and what they really need in the way of Applewriter program support and improvements.

This volume and its companion disks form a compendium of everything that has been asked for, ranging from bare beginner questions through full source code capturing and full expansion for gonzo hackers.

Thorough and extensive coverage is included for five different versions of Applewriter. Two of these versions involve older Applewriter IIe. They are called OBJ.APWRT[E (64K IIe) and OBJ.APWRT][F (128K IIe and IIc). Three of the versions involve new ProDOS® Applewriter 2.0. They are called AWB.SYS (40 column IIc only), AWC.SYS (64K IIe only), and AWD.SYS (128K IIe and 80 column IIc). Both "new" and "old" IIe monitor versions are also supported. If you are into Applewriter at all, you are bound to find something here for you, such as:

- Answers to the most-asked Applewriter questions
- Patches for null, shortline, Grappler® and others
- Self prompting glossaries for major printers
- Microjustification and proportional spacing routines
- Camera-ready print quality secrets
- Complete and thorough disassembly script
- Source-code capturing instructions
- WPL routines for columns, space-on-disk, etc.
- Information concerning continuing support, helpline, and upgrades

The first two chapters handle answers to just about everything that anyone has asked for on the helpline. Here you will find the most needed NULL, shortline, expansion, and IIc detrashing patches, details on the magic of .pd8, WPL routines to "rearrange" daisywheel spokes, ways to

simplify mailing list management, ways to list the unlistable, and even ways to print bullets.

The third chapter describes the secrets of top print quality. Included are a WPL CAMERA READY routine that will give you outstanding hard copy, sane sources of supplies, techniques of aligning proportional columns, and even that old WD40 ribbon ploy.

Next, in Chapter 4, come microjustification tips and proportional spacing secrets. Believe it or not, you can actually do these things invisibly and automatically, even on your already existing text files. Special spacing does, however, require a printer with a fair amount of smarts. We use a Diablo™ 630 because its print quality is unsurpassed, and it has its own internal microjustification routines.

Few people realize that making a glossary self prompting is very easy. Chapter 5 tells all, including how to make glossaries with instant, built in help screens and examples of complete self prompting glossaries for four of the most popular printer families. No more forgotten commands, lost lists, or sloppy stuff taped on your Apple.

Machine language hackers will appreciate the patches in Chapter 6 that give hackers ways of improving, linking, and expanding the Applewriter code. Included are two sets of patches—one set of eight for Applewriter IIe and a separate set of eleven for new ProDOS Applewriter 2.0, the Grappler printer card fix, and a prefix "dehassler."

Chapter 7 follows with a most thorough and complete disassembly script. The exact details of each and every internal ProDOS 2.0 module and entry point are covered, so you can find out exactly how Applewriter works and how to modify the program.

Chapter 8 contains full details on capturing your own ProDOS 2.0 source code and a neat WPL routine that lets you format justified text into two or more columns.

Complementing the book are a helpline service and a pair of companion disks. The companion helpline will try and answer any and all Applewriter problems you may have.

The companion disks are ready to run and hold everything you need. Included are some bonus programs that will do an author's automatic keyword indexing, a WPL program-length extender, plus a few other goodies.

I also have work in progress involving HIRES dumps, speech synthesizer links for the handicapped, code extensions, Appleworks™ compatibility, microjustification for both the Imagewriter and especially the laser printer, and several other neat goodies. Again, call us on the helpline for information.

Disclaimer time: Apple®, Applewriter, and Applesoft® are registered trademarks of some obscure outfit out in California, and you know how those Californians are. Neither Apple Computer, Inc. nor the original program author has in any way endorsed or approved of the stuff you read here. What you find inside is mostly mine, and every bit of it is fully independent.



Much of this book will apply to most Applewriter versions. I tried to make any exceptions fairly obvious. Although every attempt has been made to be useful and accurate, the only guarantee I will make is "approximate quantity one". I will attempt to correct and improve things as best I can. Be sure to make any patches or improvements *only* on your third or higher backup copies.

Don Lancaster

*This book is dedicated to the philosophy  
that no matter where you go—there you are.*



---

# 1

---

## **Applewriter Answers from the Gila Helpline**

Solid and useful solutions  
to the most asked about Applewriter problems.  
Here you will find  
cures for the NULL,  
shortline,  
the IIc trashing hassles,  
details on custom mods,  
and loads more . . .

---



The Gila Valley Apple Growers Association has been maintaining a voice helpline for Applewriter users at (602) 428-4073. The service is free except for the usual phone charges. Preferred calling times are 8 to 5 (weekdays) Mountain Standard Time. As you might expect, after a few months of operation, the same questions were asked again and again. Gathered together here are the most asked about and most needed solutions to Applewriter related problems.

One confusing point: Two wildly different Applewriter versions have nearly identical names! *Old* Applewriter 2.0 is intended only for the II and II+. This program is hopelessly klutzy, crippled, and obsolete. Unfortunately, it is the only stock program (except for some really old junk) that runs on a II+, a Franklin, or a Hong Kong knockoff. We will always use the prefix "old" when and if we describe this code.

The next newer version is called Applewriter IIe and is intended only for the IIe. We will abbreviate this one to AWIIe. Stock Applewriter IIe will not display attractively on a IIc or a "new" ROM IIe unless a simple patch is made to the program.

The latest, and by far the best, version is ProDOS Applewriter 2.0. We will always use the word *ProDOS* when talking about this version, which runs on a IIc or an "old" or "new" IIe. It does *not* run on a II+ or a clone.

In this book, we use the "WPL method" of showing control commands. Thus, [S] means *hold down the control key, press and release the S key, then release the control key*.

Ready? Here goes . . .

## ***How Can I See Exactly What Will Be Printed?***

That depends on the version. Let's start with Applewriter IIe. By changing your print destination to .pd0, you will enter a preview mode where what you see is exactly what you get when the computer prints. An [S] will temporarily stop or resume scrolling, but an [esc] will abort the print-to-screen mode.

As an alternative, you can change your print destination to `.pd8`, which will *print* a fully formatted text to your disk. By loading this formatted document, you can stay in a what-you-see-is-what-you-get mode. A formatted file can then be printed using wide open print constants, such as `.lm0`, `.rm200`, `.tm0`, `.bm0`, `.pm0`, `.tl`, `.bl`, `.ut`, `.pl66`, `.pi66`, etc. You can also force your own left margins with `[tab]` and your own right margins with `[return]`.

The intended way of using Applewriter with imbedded printing commands is far more powerful, far more flexible, far more compact, and far more useful for everything but the simplest or the most oddball of tasks.

With ProDOS Applewriter 2.0, you can independently set your left and right screen margins up to 240 characters wide. You can get an almost perfect what-you-see-is-what-you-get by temporarily setting your right margin to the *difference* between your normal printed right and left margins and by using `[tab]` rather than paragraph margins.

## ***Tell Me More About .pd8***

The command `.pd8` instructs Applewriter to *print* to your disk rather than to a printer or screen. The document, exactly the way it will appear on paper and stripped of *all* imbedded commands, goes on a file on disk. A formatted text file on disk is particularly useful for typesetting, telecommunications, or passing some text file on to another computer. This formatted file is also handy for advanced tricks like multiple columns, multiple headers or footers, custom post processing, or camera-ready print tricks.

When you use `.pd8`, be sure to add a new filename or a *.formatted* or *.fmt* trailer. Also note that formatted files are longer than normal ones because all the left margin and paragraph margins are padded with spaces.

## ***How Do I Print One Middle Page of a Long File?***

An imbedded print command called `.ep` (short for *enable printer*) is available to print single pages in mid-file. An `.ep0` suppresses printing. An `.ep1` enables printing. To print page 13 of a 22-page document, put an `.ep0` at the beginning of the text file, an `.ep1` at the end of page 12, an `.ep0` at the end of page 13, then print.

## ***How Do I Imbed NULLs?***

It depends on your version of Applewriter.

On *old* Applewriter 2.0, a NULL was imbedded with a `[V]@[V]` command. On stock Applewriter IIe, NULLs are disallowed. NULLs are

particularly important for providing underlining and superscripting on older Epson™ and other dot matrix printers. The simplest and cleanest Applewriter IIe solution is to make a two byte correction to your word processing program.

Program A.1 is an Applesoft routine called the AWIIe NULLIFIER. Note that programs described in this book are listed in Appendix A. You make a third backup copy of AWIIe under DOS 3.3e, run this repair program, and your third or higher AWIIe backups will automatically let you imbed NULLs anywhere you want them. NULLs are imbedded with a `[V][@][V]` or the easier to type `[V][2][V]`. Obviously, you can include NULLs in your printer glossary for one key access.

Do not make this modification on either of your original disks. Change only your third or higher backup copies.

Several minor gotchas: You lose any ability to run cardless 40 column text, and you must not imbed a NULL into a WPL label. You also lose an obscure use of [delete], but this key is deadly and should never be used anyhow.

The NULLIFIER adds a minor bug to the case changer. If you exit the case changer by hitting a space, you may add a NULL to your text file. The cure is to always exit the case changer by pressing [↑] followed by the [↓].

Note that this mod is intended for Applewriter IIe only. The ready-to-run program is available on the DOS 3.3e companion disk.

On new ProDOS Applewriter 2.0, NULLs are not allowed in the text file.

Period.

Instead, the stock code will automatically substitute a NULL for each US user separator found. A user separator is keyed as [—].

Although [—] substitution solves the NULL problem, it causes troubles for users needing user separators to, ferinstance, control a modem or handle the HMI daisywheel horizontal motion commands.

See Chapter 6 for a patch to redefine the ProDOS 2.0 NULL character any way you like.

## ***How Do I Produce Underlining and Superscripting on an Epson?***

You imbed the escape commands and NULL characters as needed by your printer.

On AWIIe, you make the above code modifications. On ProDOS 2.0, you substitute a user separator [—] any time you need a NULL. A fully automatic and self-prompting Epson glossary named EGLOSS appears in Chapter 5. EGLOSS is easily adapted to most Epson-like printers.

An AGLOSS for the Apple Letter Quality Printer, a DGLOSS for the Diablo and an IGLOSS for the Imagewriter are also shown in Chapter 5. These self-prompting glossaries are provided on both companion disks,

ready for your immediate use. The LGLOSS and PGLOSS for the Laserwriter™ is also available. Contact the helpline for more information.

## ***How Do I Produce Superscripting on an Apple DMP?***

The Apple dot matrix printer needs a special character set loaded before the printer will superscript.

This loading is easily done with a preboot disk. Although the Gila helpline does not currently support this preboot disk, it is in the public domain and is available, among other places, from Minuteware, Box 2392, Columbia MD, 21045, (301) 995-1166.

## ***How Do I Solve the Shortline Problem?***

The shortline problem is caused by Applewriter counting imbedded printer commands as legal characters.

If you imbed printer commands while in the fill justify mode, each line with an imbedded command will get shorter. For instance, an imbedded underline command will typically shorten that line by four characters.

Program A.2 is another Applesoft program named the AWIIe STRETCHIFIER that will automatically test and then modify your third or higher backup copy of AWIIe. For every [esc] found in any line, that line is temporarily lengthened by two characters. Should word wraparound be needed, any escapes beyond the last whole word are automatically ignored.

(A patch that accomplishes the same thing as STRETCHIFIER for ProDOS Applewriter 2.0 is described in Chapter 6 and is listed in Appendix B.) The result of STRETCHIFIER is an exact AWIIe fix for imbedded escape commands followed by a single letter. This program is available ready to run on both companion disks.

## ***What About Multicharacter Imbedded Escape Commands?***

The trick here is to use one of the above STRETCHIFIER programs and "bank" as many characters as you think you will need for the imbeddings.

For instance, an [esc][esc] should bank two characters for you, whereas an [esc][@] will be ignored by most printers but will bank a single character for you.



This approach makes more sense than trying to filter each multicharacter Escape command for each and every printer. The process can be automated by putting the banking ahead of the needed multicharacter command in your printer glossary.

## ***How Do I Improve Underlining?***

Let your printer do the underlining for you. Imbed underline commands when and where needed. Depending on your printer, the NULLIFIER and STRETCHIFIER may be needed to imbed commands. A glossary for your printer is also most helpful. All of the problems involved in underlining up to punctuation magically disappear when the printer does its own underlining. In addition, your underlining most likely will end up more uniform and darker.

## ***How Do I Make Backup Copies?***

On Applewriter IIe, all the usual advanced bit copier methods work. Copy II +™ with a parameter change of 10:96 does the job nicely.

ProDOS Applewriter 2.0 is fully copyable and unlocked. Easily copied with the Filer or any other ProDOS based copy utility, ProDOS is also easily installed on any hard disk system of your choice without any access hassles.

## ***What Is the Grappler Problem?***

ProDOS Applewriter 2.0 is intended primarily for the stock IIc Apple serial interface. Many combinations of parallel printer cards and printers behave erratically and thus require custom and individualized patches.

The typical Grappler card symptom is a random burst of 22 spaces inserted every 240 characters, which is easily patched. Details appear in Chapter 6.

The usual card problem is that so-called "intelligent" printer cards assume something is happening to page zero location \$24 and that video echo does in fact reach the screen routines. Neither is the case with ProDOS Applewriter 2.0.

## ***Can I Link an Assembler to Applewriter?***

You bet.

Apple's own newly upgraded and overhauled EDASM lends itself beautifully to editing under AWIIe. Full details appear in my book, *Assembler Cookbook* (SAMS #22331). WPL routines can automatically handle numbering, renumbering, and tabbing.

## ***Can I Process Pictures Rather Than Words?***

Absolutely.

In fact, AWIIe is even better at processing pictures than it is at processing words. You see, plotter commands are nothing but long strings of text characters, and WPL is among the most powerful ways known to manipulate long strings of text characters under programmable control. The Hewlett Packard 7470 plotter is particularly well suited for picture processing under Applewriter IIe.

With Applewriter linked to the Laserwriter, it is now possible to do mixed text and graphics that totally outperform *anything* done on the Mac. Write or call the helpline for a free demo pack that gives conclusive proof of this.

## ***How Can I Make My Own Patches?***

Program A.3 is yet another Applesoft program called the AWIIe PATCHIFIER. This program will automatically modify your third or higher backup copy of AWIIe. The program works by converting the [O]-C option from "Verify File" to "Blood Patch".

Among its many other uses, PATCHIFIER gives you a roundabout but highly useful PEEK and POKE capability. This capability either can be used directly or under WPL. Two possible uses include linking the code for a HIRES screen dump or scanning a plotter that has had its pen replaced with a photocell for automated image-to-text conversions. The AWIIe PATCHIFIER is available ready-to-run on the companion disk.

A *very* important gotcha: POKE can kill!

Any changes that you make to the code with your own patches can destroy the integrity of the entire program! *Never* save a patch to your original AWIIe disks! Always use your third or higher backup copy.

Patching ProDOS Applewriter 2.0 is much trickier because of the ProDOS operating system and the need to use machine language interface MLI links. The best route here is by way of a general expansion module. Call the helpline for more details.

## ***Show Me a PATCHIFIER Example***

The [Q]-K quit command is useless in AWIIe because you always have to reboot anyhow.

You can easily divert this command to your own needs. The CURSIFIER patch in Chapter 6 will divert [Q]-K so that it will

automatically put the cursed character into the WPL \$D string. If you are at all into WPL, you will find this sorely needed and most useful. The CURSIFIER eliminates a long song and dance involved in testing the character presently being looked at.

An improved ProDOS Applewriter 2.0 CURSIFIER patch, among many others, is described in Chapter 6. The CURSIFIER and other patches are printed in Appendix B.

## ***What Trashes the IIc Status Display?***

If you try running AWIIe on a "new" IIe or a IIc, you will see symbol and filename errors on the status line, along with an occasional and temporary change of the flashing cursor to a new symbol.

These errors are caused by a mouse nest in the IIc character generator. On the IIe, inverse uppercase characters can be in either of two code ranges, hex \$00-1F or \$40-5F. On the IIc, the \$40-5F range is reserved for the mouse text characters. Older AWIIe programs use the \$40-5F range for inverse uppercase display and thus get trashed on a IIc.

Program A.4 is the AWIIe CLARIFIER that improves the IIc status display. CLARIFIER works by remapping the inverse uppercase status line characters into a range that is compatible both with the old and new IIe as well as the IIc.

That rare and temporary change in the new IIe or IIc cursor to a new symbol is caused by parking the cursor on an uppercase character. The regular cursor should come back when you move anywhere else. This bug is minor, sort of cute, and not worth fixing.

A new ROM pair is now available for the IIe that is similar to the IIc ROMs. The CLARIFIER will be needed here also.

ProDOS Applewriter 2.0 is fully IIc and new IIe compatible without any need for detrashing programs.

## ***How Can I Print a Line of Dots?***

AWIIe ignores any line that starts with a carriage return followed by a period.

If your first character on a line is a period, that line will not be printed. To print a line of dots, start your line with one or more spaces.

## ***How Do I Make 64 Identical Labels?***

The solution to this is fun. Do one label. Then do a [L]# six times. The [L]# command will copy *all* of memory to memory! Your single label becomes two, then four, then eight, then 16, then 32, and finally 64. If you get carried away, you will eventually overflow the computer's memory.

## What Copy Options Do I Have?

The options you have depend on the length of copy that you want to move.

If the copy is less than 1024 characters, set [D] to <, hold down [⌘] and press [W] or [X] as often as needed. This process will put copies of each word or paragraph in a saving buffer.

To complete the copy, move the cursor to your new place, reverse the [D] data direction, then dump your copy with repeated [W] or [X] commands. Note that a [W] or an [X] by itself moves text. Press the same keys while holding down [closed apple] and the program copies text instead of moving it.

Another method is to do a memory to memory load while using delimiters. Note the starting and ending strings to be moved. These strings must be long enough so that they are unique. Enter [L]#/startstring/endstring/ where you want the text to appear. This process works on any length text. Should you *not* want the text markers, enter [L]#/startstring/endstring/N.

With ProDOS Applewriter, be sure to use exclamation points, rather than slashes, as delimiters.

Another way to copy is to save a module to disk, then reload the module in the intended place. This method is best for boilerplate that you may want to reuse elsewhere in another text file.

On ProDOS Applewriter 2.0, partial disk saves are much easier. To save a part of memory to disk, park the cursor at the beginning of the piece to be saved or moved, then add an ending delimiter to your filename.

## What Trashes My Text Files?

Under DOS 3.3e, you are not allowed to use any punctuation in any AWIIe filename, except for spaces and periods. All other punctuation is reserved for use by the powerful AWIIe search and replace commands. For instance, if the filename being loaded contains three dashes, AWIIe will read the real filename as everything up to the first dash. The computer then will search for the starting string between the first and second dashes and for the ending string between the second and third dashes. As a result, the computer will display a FILE NOT FOUND or may give you a partial load.

Renaming your files can create this problem, so be careful. Keep all punctuation out of your filenames at all times.

On ProDOS Applewriter 2.0, the filenames are even more restrictive. You are only allowed a maximum of sixteen letters, numbers, or periods. Punctuation of any kind, including spaces, is a no-no.

## ***What if the Text File Really Is Sick?***

Your solution depends on exactly what the problem is. If the text file will catalog but not load, first check to be sure you did not spell the filename wrong. Be very careful not to mix up "eyes," "ells," and "ones" or "ohs" and "zeros."

If the filename is spelled correctly, find out the *exact* filename in the DOS directory. Hidden control characters or other surprises may be lurking in the filename. Use the fancy catalog options in Copy II+ or read the directory tracks with a suitable disk-snoop program, starting with DOS 3.3e Track \$11, Sector \$0F. The books *Beneath Apple DOS* and *Beneath Apple ProDOS* are absolutely essential for this sort of thing.

If you find errors in the directory, you can often zap them into a useful filename. If all else fails, make a bit copy of your disk, then try to fix the initialization and the directory by using a repair program such as Quality Software's Bag of Tricks. Make the repairs only on your new bit copy.

Both Applewriter versions are extremely robust and will rarely, if ever, damage a disk by themselves. The usual causes of blowups are dirty or loose card and cable contacts, improper filenames, power line problems, or a IIe that has been overstuffed with flaky cards.

Be careful when using CONVERT to update older Applewriter files. You can end up with several files with identical names if the files are longer than 16 characters.

## ***How Can I Put a Catalog in My Text File?***

You are not reading your manuals! A simple [O]A# does the job. The # trailer says to load to memory rather than directly to the screen.

## ***How Can I Improve the Catalog Display?***

The "user" solution is to create a WPL program to do your catalog for you, including the # catalog to work file option. Then eliminate two out of three carriage returns in the catalog area to give you a triple column catalog that shows everything on one screen. Other catalog bells and whistles can be added any way you like. Be sure to include an erase option to remove the catalog from the work file when you are finished.

The "hacker" solution is to rearrange the code to suit yourself.

## ***Can I Run a WPL Program While in Mid-Print?***

You can imbed the following dot commands in your document. The commands `.lm`, `.pm`, `.rm`, `.tm`, `.bm`, `.pn`, `.pl`, `.pi`, `.li`, `.sp`, `.pd`, `.sx`, `.sy`, `.sz`, `.cr`, `.ut`, `.fj`, `.lj`, `.rj`, and `.cj` will immediately put a new value into the print constants file.

The commands `.go`, `.do`, `.qt`, `.np`, `.cp`, `.tl`, `.bl`, `.ff`, `.in`, `.pr`, `.as`, `.yd`, `.nd`, `.sr`, `.rt`, `.sc`, `.sl`, and `.ep` will immediately execute their respective code modules.

Printing stops when a carriage return is received followed by a dot followed by a legal two letter command. The two letter command is then executed. Everything on the line following the dot up to the next carriage return is treated as a passing parameter and will not be printed.

Unfortunately, the `.do` command only loads a WPL program and then sets some flags. If you imbed a `.do` command into your text file, the entire text file prints, then the WPL program executes. If you imbed several `.do` commands in your file, the entire file prints, then only the *last* WPL program executes.

Thus, in stock Applewriter, you can print while running WPL, but you cannot run WPL while printing. Hackers can trap the code on the way to the dot interpreter. This is one heavyweight solution to easy HIRES dumping in mid-document.

## ***How Can I Search or Replace Backspace Commands?***

The backspace and frontspace commands cannot be included in a stock search or search and replace string.

Instead, you can get these commands out of a glossary or by using modified WPL to individually test each cursed character with the CURSIFIER. The simplest route is to manually hand correct, using `[V][H][V]` when and as needed.

## ***Why Can't I Print a ( < ?***

Because a ( < is reserved to start footnote strings.

You will get a footnote overflow if this combination ever crops up. One solution is to imbed a space and backspace if you ever have to print this oddball combination. Another is to use a BOLD PS daisywheel and spoke rearrangement.

## ***How Can I Improve Daisywheel Underlining?***

If you use the Applewriter underliner on a proportionally spaced daisywheel printer, the spacing values get messed up on each of the backspaces used to precede an underline. This will crowd some characters and stretch others. The solution is to imbed print commands that use the printer's underliner instead of Applewriter's.

In Chapters 3 and 4, we will see several good ways to dramatically improve underlining quality.

## ***How Can I Improve My Daisywheel Print Quality?***

First, use a proportionally spaced metal daisywheel element and a film ribbon. Secondly, we will see in Chapter 4 how to microjustify, proportionally space, and dramatically upgrade your print quality to camera-ready status.

Given enough sneakiness, just about anything can be converted into a text file, including Applesoft programs, hex dumps, disassembly listings, assembler printouts, disk catalogs, WPL code, source code, glossaries, modem downloads, or virtually anything you look at or see on paper. Once in a text file, you can use the techniques of Chapter 4 to upgrade or improve the appearance any way you like.

## ***How Do I Do Kerning?***

*Kerning* is the proportional spacing of characters so that they look more natural together. How you kern depends on the printer that you are using. On a Diablo 630 daisywheel printer, a command of `[esc][H]` will back you up by 1/120th of an inch, and `[esc][Q][A]` will stretch characters out by the ASCII value of the last imbedded command. Thus [A] gets you 1/120th of an inch, [B] gives you 2/120ths, and so on. Kerning is reset with an `[esc][Q]@`. Note that the at sign (@) is an ordinary character and *not* a control command.

A kerning feature is included in the self-prompting DGLOSS glossary of Chapter 5.

## ***How Can I Put Comments in the Glossary?***

Start your comment line with a question mark (?), a slash (/), or an asterisk (\*).

These are the three disallowed glossary cue characters. To enter your comment, use the main editor rather than the [G]-? glossary option. See Chapter 5 for more on glossaries.

## ***Why Do Some Glossary Strings Execute Immediately?***

Unlike old Applewriter 2.0, both the Applewriter IIe and ProDOS Applewriter 2.0 will execute any imbedded string control character exactly as WPL does.

If you want your glossary to imbed a control character into your text, precede *and* follow that control character with a [V]. Thus, to imbed an [esc], you should put a [V][esc][V] into the glossary string. *Only* the [esc] goes into the actual text file.

Many of the other Applewriter books miss this key point: The glossary action of older Applewriter 2.0 is wildly and totally different from the glossary action of AWIIe or ProDOS 2.0. Thus, most very old glossaries may not be compatible with modern code.

## ***How Do I Imbed a [V] in a File?***

With AWIIe or ProDOS 2.0, the Verbatim command [V] can be entered directly into the glossary or put directly into a find string. To imbed a [V] in a text file, use either a search and replace under [F] or a replacement under [G]. Note that using search and replace is much simpler than the roundabout method once needed to imbed [V] under old Applewriter 2.0.

## ***How Can I Fake Superscripting or Subscripting?***

If your printer does not have specific commands for superscripting or subscripting, see whether you can do a negative halfline feed or a positive halfline feed. To superscript, first imbed a negative halfline feed, type the characters to be superscripted, then follow with a positive half linefeed. Reverse the process for subscripting. Either method will print fullsize characters that are offset half a line above or below normal.

Note that the Diablo people use an [esc]-U to move *down* the page and an [esc]-D to move *up* the page. You also might be able to use a graphics mode to move the paper, return to text and enter the characters to be superscripted or subscripted, go back to graphics to move the paper back to its original line, and return to typing normal text.



## ***How Do I Print Several Columns?***

Listing C.15 shows you a two column routine that easily extended to three or nine columns. See Chapter 8 for more details.

## ***How Do I Improve Daisywheel Registration?***

Registration can be a problem in graphics, in multicolumn printing, and any time you want to back up on the paper.

First, use a bidirectional tractor instead of a unidirectional one. Second, see if the paper-feed stepper has a backlash adjustment. On the Diablo 630, the screwdriver backlash adjustment is in the center of a three-inch black gear just under the right-hand platen knob. Watch your adjustment range: Too tight binds and causes excessive wear. Too loose misregisters. Most newer daisywheels include automatic antibacklash gearing.

As a sledgehammer solution to fix misregistration of the top line of a new column, try this: Back the printer up even farther than you really wanted, print a single period, then go forward a line or two and start printing. This technique should align the top printed line with previous printing. When this is all done, either erase or ignore the period.

## ***How Do I Load a WPL String from Text?***

All of the usual load commands work with pls. To load a string from memory that starts with *aardvark* and ends with *zebra*, just enter `pls#/aardvark/zebra/= $A`.

Several gotchas here: The loaded string may be no longer than 64 characters, and the delimiters must be unique. If you do not want the delimiters to appear, an N trailer is allowed. If you do not know the string you want to load, use a search and replace to identify its start with a unique character. Such use is common with mailing lists.

If you are within 64 characters of memory overflow, you may not be able to load a WPL string.

A sledgehammer solution is to use either CURSIFIER patch (Patch B.3 for AWIIe or Patch B.18 for ProDOS) described in Chapter 6. These patches let you scan the document one character at a time.



---

# 2

---

## **More Applewriter Answers from the Gila Helpline**

Solving the page creep problem,  
rearranging the spokes  
on a daisywheel printer,  
mailing list and form letter hints,  
easing ProDOS prefix hassles,  
plus some truly wondrous  
new WPL routines . . .

---



## *How Do You Make a Glossary Self Prompting?*

Any control commands that are imbedded in a glossary will execute immediately as if they were WPL instructions.

So you could put your tutorial or prompting message into the glossary, using ] fake carriage returns as needed to separate lines. On cue, the tutorial or message that you specified would appear on the screen.

This tutorial method for glossaries has two disadvantages. First, full screens will take too long to display. Secondly, your message gets tacked to your work file and has to be removed later.

A sneakier tutorial method exists and solves both problems.

On cue, load your glossary tutorial or prompt directly *from* the disk and *to* the screen, a process that takes no longer than three seconds and leaves your work files undisturbed. For instance, after all your regular glossary entries are completed, enter a full help screen. Suppose this screen starts with *Once upon a time* and ends with *happily ever after*. Suppose also that your glossary is called MYGLOSS. The glossary entry *z [L] MYGLOSS/Once upon/ever after/* will automatically load your glossary-based tutorial to screen only, keyed on an [G]-z.

One very important gotcha: Be absolutely certain that the tutorial appears in the glossary *before* the *z* entry! Otherwise the tutorial Find command will find *itself* rather than the message that this command is supposed to put on the screen.

Here's a real heavy: The glossary entry of *Z [Q] FMYFLOSS [L] MYGLOSS/Once upon/ever after/* will key on an [open apple]-Z to both save your glossary to a new disk and give you the tutorial screen. Thus you would use [open apple]-z for just the tutorial and [open apple]-Z for both a tutorial and a disk save.

We will be seeing much more on glossary entries in Chapter 5.

## How Do I Solve the ProDOS 2.0 Prefix Hassle?

Any time that you boot or change disks, you must change the ProDOS prefix or you will get an error message.

The hard way to handle prefixes is to memorize the prefix name on every disk and use [O]-H to reset *each* disk on *every* change. Several easier ways to handle prefixes exist. One way is to use ,d1 or ,d2 as prefix names. These names automatically set the prefix to the name of whatever happens to be in the drive at the time the prefix is set. Setting the prefix to what is in the drive is far simpler and easier than memorizing names.

Better yet, here are some sledgehammer solutions that completely eliminate most prefix hassles.

First, create a STARTUP program that includes a WPL line of *[space] OH,D2 [return]* and save this program to your booting disk. During a cold boot, STARTUP will automatically set the prefix to whatever you have in the second drive. Secondly, create two entries on your favorite glossary of *1[O]H,D1[O]A* and *2[O]H,D2,[O]A*. Note that *pairs* of brackets here enclose a Control command and that single closing brackets substitute for carriage returns.

Any time that you have to change disks or drives, use *[closed apple] 1* to automatically set the new disk's prefix to drive one. Pressing *[open apple] 2* will set the prefix for drive two.

See Chapter 6 for more details on setting prefixes.

## Why Do Often Used Glossaries Get Longer?

Every time you resave a glossary to disk, one additional carriage return may be inadvertently tacked on the end.

If you have a very long glossary and repeatedly pass it from disk to disk, eventually you could overflow the glossary memory limit of 2048 characters. One solution to this problem is to use *[F]<>>><><A* before you resave.

## How Can I Print All of My Daisywheel Spokes?

The ASCII code has only 96 printable characters.

On most 96 spoke daisywheel printers, the space (\$20) and delete (\$7F) codes are reserved. Thus, you can directly print only 94 of the spokes on a 96 spoke daisywheel. You have to be sneaky to use the other two spokes.

On the TITAN 10 element, the two hidden spokes are the cents and the closing single quote symbols. On the BOLD PS element, the two hidden characters are the at sign and the right bracket. Diablo codes their spokes as `[esc]-Y` and `[esc]-Z`, respectively. To print a cents symbol on the TITAN 10 wheel, imbed an `[esc]-Y` in your work file.

## Why Does the BOLD PS Printwheel Foul Up Punctuation?

Not all daisywheel elements have exactly the symbols needed in exactly the order coded by the Apple IIe. Some spokes may have different characters. Other wheels may have the correct character but in a position different than that specified by the normal ASCII code.

On the TITAN 10 wheel, all of the spokes print normally and exactly match the Apple code. As Figure 2.1 shows, the BOLD PS wheel has 15 "problem" spokes. Seven of these spokes are coded incorrectly. The WPL.SPOKE REARRANGER of Program A.5 will cure these. The remaining eight spokes hold unusual symbols, such as a copyright, two trademarks, a double underline, a degree symbol, and a paragraph sign.

APPLE	TITAN 10	BOLD PS
[	[	©
]	]	°
{	{	¶
}	}	§
<	<	[
>	>	=
^	^	!
		!
\	\	<
/	/	>
~	~	®
@	@	†
[esc] Y	¢	™
[esc] Z	'	@
		]

Fig. 2.1. Not all daisywheel petals are the same, nor are they always in the same spoke position. Here are the key differences between two popular daisywheels.

## ***What Causes the Diablo 630 Secondline Problem?***

The Diablo 630 daisywheel contains an obscure bug that will louse up underlining, hidden-spoke access, or other imbedded commands on the first right-to-left pass following a switch to full microjustification.

This bug apparently is caused by certain parameters not being initialized properly in the enhanced 630 firmware.

The usual symptom is missed or garbled underlining. This nearly always happens on the second line of each paragraph. An extra character also may appear at the extreme left margin. The line may be longer or staggered from its intended position as well. One sledgehammer cure is to print those paragraphs that contain *any* underlining or funny spoke access left-to-right only.

See Chapter 3 for more details on second line problems.

## ***What Is Meant by the Words Diablo Compatible?***

A Diablo compatible printer has the prongs on its power cord the same size and spacing as that on a Diablo 630. It will thus fit in the same AC wall outlet with a minimum of excessive force.

The bare *minimum* requirements for Diablo compatibility are a full firmware microjustification, full HyPlot vector graphics, the full and total imbedded command capability including *all* word processing and *all* diagnostic options, ability to use plastic or metal printwheels of various number of spokes interchangeably, non-volatile print parameter saves, and the ability to run forever 24 hours a day without any degradation of outstanding print quality.

## ***How Can I Proportionally Space and Microjustify?***

You proportionally space and microjustify by imbedding commands in your work files that "instruct" your printer to handle these tasks for you.

To do so, a printer must have a fair amount of smarts. It's best when the printer has its own internal microjustification firmware. Chapter 3 includes an automatic and invisible proportional spacing and microjustification package.



## ***What Difference Does Extended Memory Make?***

Two separate Applewriter IIe programs exist on the boot diskette. If you do not have extended memory, OBJ.APWRT][E is booted, which gives you a workspace of 27,645 characters. If you have a IIc or IIe with extended memory, OBJ.APWRT][F is booted, which gives you a much larger workspace of 48,845 characters. If you make any patches or changes to the code, remember to patch *only* the version that you are using. Patches to the E code version will clobber F code and vice versa.

ProDOS Applewriter 2.0 has three separate programs. AWB.SYS gives you a 48K text file on a IIc in its 40 column mode. AWC.SYS on a short 64K IIe cripples your computer's potential with a meager 22K text file. AWD.SYS gives you a 48K text file on a IIc in its 80 column mode or on either "old" or "new" IIe with extended 128K memory.

## ***What Programs Must I Keep on the Boot Disk?***

You should *never* erase *any* program from either your Applewriter master disk or its factory-supplied backup. Make changes only to your third or higher backup copy.

On Applewriter IIe, you *must* keep the booting code OBJ.BOOT. If you do not have extended memory, you must keep OBJ.APWRT][E. If you have and use extended memory, keep OBJ.APWRT][F. You probably will want to keep the help screens, so keep all files that begin with an HE prefix.

Although you can safely delete PRT.SYS and TAB.SYS, doing so will give you an error message on bootup and will enter useless print and tab values in your files. It makes more sense to replace PRT.SYS and TAB.SYS with values that you often use. All remaining files can be deleted from your third or higher backup copy.

On ProDOS Applewriter 2.0, you must keep AW.SYSTEM and your choice of AWB.SYS, AWC.SYS, or AWD.SYS.

## ***What Is Special About Applewriter IIe DOS?***

The DOS 3.3e used in AWIIe is special, but the combination generates standard text files totally compatible with stock DOS 3.3e.

Specifically, the DOS is installed in high main RAM rather than its usual \$9600 BFFF location. The DOS is sometimes accessed by a custom

RWTS routine that allows easy scanning of text files for delimiters. Only two DOS files may be open at one time. The Open command is modified so that it sometimes does not create a new and nonexistent text file when reading. This saves having to delete files caused by fumble fingered typing.

The init is only an init and does not put a DOS image onto the new disk, nor does init save a HELLO program. Instead of HELLO, the boot program is named OBJ.BOOT.

The AWIIe program also attempts to write a DOS clone to auxiliary high RAM. The program fails at this because the program incorrectly "assumes" that auxiliary high RAM switches with auxiliary main RAM. Many more details appear in my *Enhancing Your Apple II*, Volume II (SAMS #21425), that really tears into the AWIIe program. The similar treatment for ProDOS Applewriter 2.0 is described in Chapter 7.

## *How Can I View the Program?*

To analyze or modify either version of Applewriter IIe, boot Dos 3.3e, then do a *BLOAD OBJ.APWRT]/[F,A\$2300* for the *extended* memory version. Note that a stock BLOAD will put the F program in the wrong place in memory. Both E and F programs are intended to be loaded and run starting at \$2300.

To analyze or modify any of the three versions of ProDOS Applewriter 2.0, do a *BLOAD AWD.SYS, A\$2000, TSYs*. Note that the Type command TSYs says to load a system file as if it were a binary image.

## *How Do I Convert My Binary Files to Text Files?*

Huh?

You must still be using old, old Applewriter 1.0 or 1.1. These dudes used oddball binary files instead of stock text files. All newer Applewriter versions use standard ASCII text files. Applewriter IIe has a provision to convert old binary files for you, but ProDOS Applewriter does not. Should you need your standard text files completely stripped of all Applewriter commands, use a .pd8 print to disk with either new version.

## *What Makes the [Closed Apple] Key "Stick"?*

The apple keys are really the game paddle buttons. [open apple] is equivalent to button two and [closed apple] is equivalent to button one.



This strange looking command finds lots of "any" characters, then replaces them with nothing. This command is one that is easily put in your favorite glossary.

## ***How Can I Imbed Hidden Lines in a Mailing List?***

Hidden lines are useful to hold order records, expiration dates, and stuff like that.

Simply start all your hidden lines with two periods. You can then code your hidden lines any way you want. A line starting with one period will not be printed. A line starting with two periods will neither be printed nor mistaken for an imbedded command.

## ***So How Do I Print the Hidden Lines on a Mailing List?***

Any line starting with one or more periods will not show up on your mailing labels. For a directory or other printed record, use the `[F] < > .. < > .. <` command to replace every carriage return that is followed by a double period with a carriage return, a following space, and a double period. To hide the lines again, do a `[F] < > .. < > .. < A`.

To avoid mixups, it pays to reveal hidden lines only on screen and never on disk.

## ***How Do I Add a Common Header to a Mailing List?***

Suppose that you want to address the Editorial Department of each magazine on a list. Suppose further that each address on the list is separated by three carriage returns. If you do a `[F] < > > > < > > Editorial Department > < A`, you will replace the blank line before each address with the needed header.

## ***How Do I Get the Page Numbers to Print?***

Read your manuals.

Any place in a header or a footer that a # appears, the page number will appear. Thus `[P] TL/page #//` will give you a "page" and page number centered on the top, and `[P] BL/#//` will give you only the actual page number at *bottom left*.

## ***How Can I Print Double Headers or Footers?***

There are several ways to double headers or footers.

One is to use a .pd8 to format the document and then hand patch any custom headers.

The second and best way is to use WPL to create custom headers or footers on a page by page basis. This is best done by subtracting the number of excess header or footer lines from the 66 line page length and setting .pl for the new number, then using .pd8. Key the WPL on some unique marker.

You can also use WPL to do multinumbered headers, such as *Chapter VI, page 19*. Just use one or more separate text files per chapter and let WPL change the top or bottom line as each new segment is loaded. The .pcp command then links the pieces together.

## ***How Can I Print a Glossary or a WPL File?***

Glossaries or WPL programs are tricky to print because they usually will contain control characters. These control characters will be ignored or will make your printer do wildly wrong things.

Program A.6 is a WPL program called WPL.FILE LISTER intended to print WPL programs or glossaries that contain imbedded control commands. Each imbedded control command is replaced with a letter and a pair of brackets, such as the screen clearing [L]. The result is an easy to view but nonworking listing.

This simple program will *not* find [H] backspaces, [U] frontspaces, or the start the footnote machine (< commands, all of which are rare. You will have to hand patch these. Note that the [F] command will not accept a backspace or a frontspace.

Ironically, WPL.FILE LISTER is not so great at listing itself. So, we have

souped up the listing by making all of the *real* control characters bolder than the viewable bracket pairs.

All the WPL routines in this chapter are available ready-to-run on the companion diskettes.

## ***How Do I Do a HIRES Dump?***

Very carefully.

The working AWIIe or ProDOS 2.0 code straddles both HIRES pages, so putting a HIRES picture where it belongs in the machine and actually viewing the picture is extremely tricky.

One solution is to precode the HIRES picture in a text file containing strings that your printer will interpret as graphics commands. Then use WPL to load and dump these strings as part of a multifile print. The precoder can be done directly in machine language or can be an Applesoft routine that links to machine language modules.

You also can use .pd8, followed by a post processing trip through Applesoft or some custom machine language rebooting.

The best solution is to modify AWIIe so that a HIRES picture can be loaded from, say, \$7000 to \$8FF8 along with a HIRES dumper that sits above or below the image. With custom mods, a simple *.hd/pixname/* will insert the desired picture right in your text, preserving page numbers and picking up the main text file undisturbed.

Note that HIRES dumps are highly printer and format dependent.

## ***How Do I Center a Title?***

The answer is once again in the fine user manuals.

You center a title by imbedding a .cj on the line before the centering is to take place.

## ***Why Won't My IIc Print Past Line 80?***

On a cold boot, the serial ports in the IIc are initialized to automatically produce a carriage return on column 80. They are also set to not echo print characters to the screen. ProDOS Applewriter automatically resets the baud rate, line width, port configuration, and video nonecho for you. This resetting is handled by entering a [O]-J.

Under AWIIe, you can set the line length of your IIc port several ways. One is to configure the port with the Utilities program before you boot Applewriter. Another is to POKE 1401, 255 for slot one or POKE 1402, 255 for slot two.

Two other routes are to imbed a *[I]255N* as the first printed line in your text file or use the sledgehammer *[I]-2*. See the *IIfc Technical Reference Manual* for full details. Once set, the line length and video nonecho is remembered only as long as power is applied.

## ***Is WPL Really That Great?***

Owning Applewriter and not thoroughly understanding and using WPL is almost as stupid as buying a Porsche just to listen to its radio.

## ***What Good Is Right Justification?***

Not much, unless you are a poet.

Right justification can be used for flush right page numbers or dates, but the stock *.tl* and *.bl* commands do a quicker and neater job. Some business letter writers might prefer having the date and signature flush right.

## ***How Do I Run Old Applewriter 2.0 on a IIfc?***

The usual problem is that the stock boot of old Applewriter 2.0 on a IIfc will give you an 80 column display only with a Sup-R Term card in slot three.

One way around this is to remove the IIfc 80 column card from the expansion slot (which is really a fancified slot three) and plug the Sup-R-Term card in its usual place.

A second solution is to tear apart the boot code and change the 80 column identification check to match the 80 column card you are going to use. To use the stock slot zero IIfc card, replace the identification check with *NOP* or *\$EA* commands. But the best solution of all is . . .

## ***How Do I Use Old Applewriter 2.0 to . . .***

Dumpy old Applewriter 2.0 absolutely does not compare with either Applewriter IIfc or ProDOS 2.0.

Among the more blatantly obvious differences are the full 80 column uppercase and lowercase display; the always there and always live cursor;

the improved tabbing, loading, and copying; the catalog-to-file feature; express cursor motions; lack of escape-itis; doing away with the "open cell"; turnstiling; much longer text files; the new print-to-disk and keyboard-to-printer options; the special apple-key services; and the many improvements and upgrades in WPL.

To these benefits, the latest ProDOS version adds 240 column wide editing, horizontally scrollable screens, a what-you-see-is-what-you-get capability, page/position display, and a built in modem.

The probable bottom line is that nobody, but nobody, has tried one of the new Applewriter versions and then gone back to the old 2.0 code.

Anytime. Ever.

No, the new versions will not run on a II or a II+, which, in itself, should tell you something. The longer you wait to make your inevitable and unavoidable Applewriter upgrade (1) the higher the cost in frustration and wasted time, (2) the more your competition will gobble you gone, and (3) the sillier you will feel.

## ***How Do I Ring the Ding-Dong from WPL?***

Just enter a `ppr[G]` for a ding-dong or a `ppr[G][G][G]` for a tweedle. Use the ding-dong for errors and the tweedle to call the operator back to the machine after a long WPL routine finishes. As a reminder, a `ppr[L]` following a `pnd` will clear the WPL screen for you.

## ***What Causes Page Creep?***

Several causes exist and several cures are available.

One cause is a seemingly benign bug in both AWIIe and ProDOS Applewriter 2.0 that adds an invisible and unwanted space to the end of *all* top and bottom lines. If you set both the printer card and Applewriter to a right margin of 80, this bug will add one or two lines to the pages where the top or bottom lines are in use.

The cures are obvious: Use a 78-character right margin inside Applewriter or set your card and printer margin to 255. A patch to fix this bug is described in Chapter 6 and Patch B.14.

A second cause is not understanding .pl and .pi print constants.

The page interval, which is the physical page length, is almost always set to 66. The .pl is the total number of printed lines, including top and bottom margins, top line, bottom line, and the body. The .pl is usually set in the 58 to 62 range. Exceptions occur when running a .pd8 in which .pl is usually set to 66 or when printing labels, filing cards, legal paper, or other forms involving oddball vertical lengths.



A third cause of page creep is a printer that has its skip-over perforation activated. Either deactivate this printer feature or change the .pi page interval to something shorter so that everything comes out even.

A fourth possibility is that your interface card is forcing its own form feeds at page bottom. Once again, reprogram the card or shorten the .pi as needed.

Other causes of page creep involve imbedded printer commands.

The line counter inside Applewriter loses track of where you are on the printed page if you feed a positive or a negative line feed, use a positive or negative halfline feed, switch to graphics, or pick something other than six lines per vertical inch. The rule is to undo what you do, such as throwing in a positive line feed for each two negative halfline feeds elsewhere on the page.

## ***How Can I Stop a Hyper [delete] Key?***

The automatic repeat is deadly when used with [delete].

Once [delete] gobbles up a character, it is gone forever. Do *not ever* use [delete] for anything. Instead, use [open apple]-→ to delete, so any lost characters can be retrieved later from the swallow buffer by using [open apple]-←.

## ***How Do I Imbed a Carriage Return in the Glossary?***

The ] symbol imbedded in the glossary will be interpreted as a fake carriage return. A poor choice of type font makes the closing bracket in the example on page 56 of the AWIIe manual look almost like a vertical line.

## ***What Good Are the Apple Keys?***

[Open apple]-? or an [open apple]-/ accesses a tutorial. [Open apple]-← deletes *one* character and saves it to the swallow buffer. [Closed apple]-→ retrieves a single character from the swallow buffer. [Closed apple] used with any other character will read the glossary and enter the glossary string in the text.

[Closed apple]-← gives you a by-word express backspace. [Closed apple]-→ gives you a by-word express frontspace. [Closed apple]-↑ gives you an express jump 12 lines back toward the beginning of your file. [Closed apple]-↓ gives you an express jump 12 lines forward toward the end of your file.

The *[closed apple]-[tab]* combination moves you forward on a line by skipping over text and without inserting spaces. This method compares to a stock tab that inserts spaces to get to the tabbed position, shoving any text in front of the cursor forward.

*[Closed apple]-W* or *[closed apple]-X* will non-destructively *copy* a word or a paragraph for relocation elsewhere. This compares to a stock *[W]* or *[X]* that *removes* the word or paragraph for relocation elsewhere. Use plain old *[W]* or *[X]* to move. Use *[closed apple]* to make a duplicate copy.

## Can I Preboot ProDOS Applewriter 2.0?

Whether you can preboot depends on what you want your preboot to do.

If you are setting printer card parameters; downloading custom typefonts; activating fancier modem functions, using a plotter; etc.; a preboot program will work just fine.

To use a preboot program, assign it as a ProDOS .SYS file type. Then put your preboot file on your third backup disk as the *first* system file type in the directory. A cold ProDOS boot will then run your preboot program. Your preboot program, in turn, should boot AW.SYSTEM.

On the other hand, note that the ProDOS Applewriter cold boot firmware very meticulously unplugs and sets aside anything that was connected to slot three. Preboots that do *anything* to a slot three card are thus doomed to failure. Thus, it is very difficult to link stock ProDOS Applewriter 2.0 to third-party video or screen cards of any type. Note that this program makes no use whatsoever of *any* monitor routines.

## Is Source Code Available?

Capturing your own source code is a simple and straightforward process. Full details for AWIIe appeared in *Enhancing Your Apple II*, Volume II (SAMS #21425). Details for ProDOS Applewriter 2.0 appear in Chapter 8.

## What Causes Missed Characters?

Using the keys for karate practice. The splintered table should have given you a clue.

Seriously, missing characters happen only to one in thirty typists. That one typist is invariably brutal in his or her key pounding.

A 64 key type-ahead buffer in Applewriter IIe makes overtyping theoretically impossible. Unfortunately, the keyscanner in the IIe is slightly flaky. You can prove this by rapidly typing on the arrow keys, rolling back and forth. Those single quotes you get illustrate the problem.

To never miss a keystroke: (1) Be gentle in your typing; (2) stay in the replace mode rather than in the insert mode as much as possible; (3) be extra crisp in your *release* of all keystrokes; and (4) touch the home keys as lightly as you possibly can. Another tip: If you must insert bunches of text in the middle of a long file, add enough carriage returns so that the text *beyond* the insertion stays temporarily off screen.

## ***What Causes Painfully Slow AWIIe Entry?***

That's a very good question.

Under certain extremely rare conditions, Applewriter IIe can get in a super-slow and virtually useless entry mode. This has happened to me only once or twice, and only one helpline call in a hundred ever mentions this problem.

Obviously, the problem is either in the hardware or the software.

If slow entry happens to you, try rebooting. If that doesn't help, try the other factory copy of the program. Next, turn the power off, remove the line cord, and carefully clean the contacts of all of your plug in cards, particularly the 80 column card, the DOS card, and your printer card. Radio Shack® cleaner-degreaser spray can be used. Then carefully lift all of your memory chips very slightly and reseat them. Raise each chip just enough to slightly scuff all of its pins.

Be sure to run your IIe memory diagnostic, using the *control[closed apple]/[reset]* that is built in the IIe monitor. If none of the preceding solves the problem, try a different Apple or change from a "short" 80 column card to an "extended" 80-column card or vice versa. Let us know what you find.

## ***Why Can't I Do a Decent Underline?***

That depends on your printer.

If your printer will accept a standard backspace as a backspace, the AWIIe underliner should work. If it does not, see whether your printer accepts only low ASCII backspaces and adjust your printer card and printer interface accordingly. The usual symptom is a character, an underline, a character, an underline, and so on. If this happens to you, your printer is not accepting high ASCII backspaces.

You can test your printer from Applesoft. Try printing a *CHR\$(08)*, then a *CHR\$(136)*. If the program buys the 8 but not the 136, you are transmitting high ASCII, and the printer is only accepting low ASCII. Often a single software word or a flipped DIP switch will solve the problem.

If the underlining looks lousy, see whether the printer has an underline command that you can imbed in your text to turn the printer's underlining on and off. See Chapters 3 and 5 for more information.

Some printers will insist on underlining the right margin or on dropping a continuing underline on the next line. Individually underlining each word is one cure. A second solution is to break the underline into a pair of underlines at the appropriate point, which can be done either by hand or with some help from .pd8.

## *How Can I Print a Solid Bullet?*

The simplest way is to use a lowercase *o* and hand ink it black. Neatness counts.

Some printers will let you download custom characters. If so, just divert some character you never use and replace it with a solid black disk. The same idea works for math symbols, pi or other greek letters, trademarks, or whatever.

On other printers, you can optionally switch to graphics, draw your bullet or whatever, then return to normal text. I like to use my Diablo 630's BOLDPS degree symbol. Even if you have to fill it in by hand, this symbol is just the right size and shape. Unfortunately, the degree symbol is a tad high of center, so I use the WPL.BULLET SHOOTER routine of Program A.7. You enter a single opening quote everywhere you want a bullet. Just before printing, run WPL.BULLET SHOOTER.

## *Is There an Easy Way to Do Form Letters?*

The factory disk includes automatic WPL form letter routines. Their use is fully detailed in the WPL manual. Beginners seem to have an inordinate amount of trouble with AUTOLETTER.

If you are willing to print only 20 letters at a time, you can do form letters a much easier way. What you do is put the form letter in your mailing list rather than vice versa.

To put the form letter in your list by hand, just enter 20 or so addresses into your machine. Then load your letter after each address. Note that your letter should *start* with the body and *end* with the date. The first date will have to be loaded by hand.

The length of the letter determines how many you can fit in the machine without an overflow. Having the 128K extended memory card makes a big difference. WPL.FLINSERT of Program A.8 is an automatic WPL form letter inserter that keys on an #### insertion marker to load a file called MYLETTER.

If you need anything fancier than this easy, quick, fast, no frills routine, use the stock AWIIe routines instead.

## ***How Do I Imbed Escape Sequences?***

This helpline call was from someone who tried using his old Applewriter 2.0 glossary on AWIIe.

To imbed a printer command of, for instance, [esc]-P to your printer, you had to use a [V][esc][esc]P [V] with old Applewriter 2.0. The first esc is not imbedded. All it does is get you in the cursor control mode. On Applewriter IIe or ProDOS Applewriter 2.0, you are always in the cursor control mode, so you need only a single escape command of [V][esc]P [V].

What happened to our caller was that two escapes were sent to his printer by his newer Applewriter version. The printer ignored an [esc][esc] as meaningless, then merrily went on to print the P as a real character.

## ***Why Do Some AWIIe Patches Disable the Help Screen?***

One of the side effects of the otherwise useless AWIIe Volume Verify code module is that it enables the help screens. If you overwrite this module with your own code, you lose the help screens. This problem is unique to AWIIe.

The Applesoft fix for the OBJ.APWRT][E version is *IF PEEK (19988) = 176 THEN POKE 19988, 182*. The similar repair for OBJ.APWRT][F is *IF PEEK (20365) = 176 THEN POKE 20365, 182*. Note that this patch is built in to the AWIIe STRETCHIFIER and AWIIe CLARIFIER, Programs A.2 and A.4. Study these programs for details.

## ***Why Does [Y] Sometimes Destroy Everything?***

If you ever mix up the turn-off-the-split-screen command of [Y]N with the flush-everything command of [N]Y, you will end up in deep trouble. Very few users make this mistake more than once.



---

# 3

---

## **Secrets of Top Quality Printing**

How to get "camera ready" print quality,  
the old WD40 ribbon ploy,  
sanely priced supplies,  
optimizing your printer interface,  
methods of custom imbedding  
printer commands,  
justifying columns,  
plus a few other goodies . . .

---





Most hard copy from most personal computers looks just plain awful. If I had my way, I would use a phototypesetter for all my computer rough drafts and quick and dirty internal printouts.

For final drafts, gravure would be nice—which says that the first person to come up with a \$400 phototypesetting or laser printing plug in for an Apple will run away with a very large bag of marbles, particularly if the font library is available for less than a dollar a whack. At this writing, the laser engines aren't quite here yet. When they do arrive, they are certain to become popular and heavily used. They are fast, silent, bit mapped for text and graphics in any mix, and can use an infinite variety of real type fonts in any size you like. Like most anyone else, I can hardly wait.

Back to reality. You are probably stuck with a dot matrix or a daisywheel printer. What can you do to be sure you have the best possible print quality on everything you send out your door? You can use some of the ideas here now no matter which printer you are using. Others are aimed at users of daisywheels, for daisywheels is where you will end up at this writing if print quality is super important to you.

We will look first at four print quality rules that apply no matter what printer you are now using. After that, I'll describe choices that can save you money and help you upgrade beyond your present print quality.

Let's do it . . .

## ***Four Print Quality Rules***

### ***Matching Final Image***

Far and away the single most important rule is . . .

### Print Quality Rule Number One

Have the original image created by the original author match the final image seen by the final reader as *exactly* as possible.

The words are not all that counts in the communication process.

How those words are arranged by the writer and how they are viewed by the reader is everything. The closer your rough draft looks like your final result, the better your print quality will be; the better you will be able to balance the content and the appearance of what you are trying to create; and, most importantly, the fewer ways people between you and the final result can foul up the works.

Now, the latter is no big deal on a business letter. But, on anything like an article, a paper, or a book, this final rule becomes crucial. Did you know that, in the traditional book publishing setup, the author *never* sees what his readers see until after the book is actually printed?

Believe it or not, neither does the review referee, the editor, the typesetter, or even the proofreader!

All of these people traditionally work with images that are *totally different* from what the reader will see. Some work with doublespaced copy, some see typeset galleys, some concentrate on elaborate commands and comments buried in the text, yet others deal with artwork proofs.

And the words and pictures are strictly and absolutely segregated throughout nearly all of the publishing process. Even during final pasteup, most of the pages are not beside one another.

What I am saying is that traditional doublespacing positively has got to go. If your editor insists on doublespacing things, show him where the vertical size control is located on the back of his video monitor. Otherwise, make what you first type look *exactly* like what you want your final reader to see. *Anything else in this day and age is not even absurd.* Remember that ProDOS Applewriter can work in a what-you-see-is-what-you-get mode by setting your screen right margin to the difference between your printed right and left margins and by using the tab key instead of paragraph margin .pm adjustments. Both ProDOS Applewriter 2.0 and AWIIe also give you WYSIWYG through your choice of (1) forcing your own carriage returns, (2) using .pd0 to print to screen, or the heavy solution of (3) which is the "heavy" solution of using .pd8 to print to disk.

## Getting All the Parts for the Printer

Our second rule is . . .

## Print Quality Rule Number Two

Be sure you get *all* the parts needed for your printer.

**Question:** A Diablo 630 printer lists for \$1800 on dealer special. How much does this printer cost you?

**Answer:** With lots of luck and some compromises, you might squeak by for less than \$3000.

All of the essential and good parts needed for top print quality are purposely left out of most of the printer packages sold by most computer dealers. First, you need the *real* manuals for your printer, which include the service, maintenance, configuration, interface, and repair manuals rather than just the vapid and cutsey whatever that got stuffed in the shipping box. Cost of these special manuals is typically \$25 to \$45 each.

Normally, a dealer goes out of his way to prevent you from ever getting your hands on any of these manuals, for each distributed copy cuts dearly into his service work. You usually have to order copies directly from the factory. These complete manuals are absolutely essential for (1) finding out how to exploit the true capabilities of your machine, (2) maintaining long term print quality, and (3) resolving compatibility hassles.

Secondly, you need some decent way of feeding paper. For daisywheels, this means a print tractor. No tractor at all is a cruel joke at best. One-way tractors simply do not hack it. A bidirectional tractor is needed for graphics and for top text quality.

Tractors are best bought used or surplus because they are far cheaper this way.

Thirdly, you need the special toolkit required to keep the printer alive. Diablo gladly lets you steal its little box that contains two funny screwdrivers, a tiny steel rod, some sticky glop, and two pieces of stamped metal for a mere \$75 plus shipping. Unfortunately, you must have these tools and parts to make any sane use of the printer at all.

Fourthly, you need to find out about the extra cost bells and whistles. Like an engine and wheels. On dot matrix printers, bells and whistles include graphics ROMs, screen dumpers, and font downloaders. On daisywheel printers, look for enhanced or expanded firmware that let you do microjustification, full word processing, vector graphics, PS table and font downloading, or other features that add intelligence.

As we'll find out later, the secret to superb print quality is to let the Applewriter software do what it does best, let the printer firmware do what it does best, and then link the two as intelligently as possible.

Finally, you probably want to build a silencer. Most any impact printer will drive you up the wall if you work anywhere nearby. Although dot

matrix printers seem somewhat quieter than daisywheels, the noise of dot matrix printers is more highly pitched and much more stressful. If you think toolkits and manuals are priced out of sight, wait till you see the pricing on silencers.

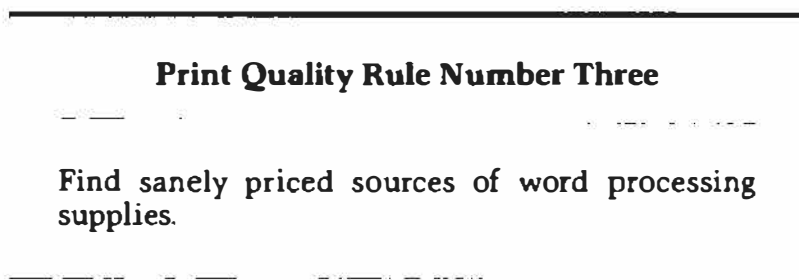
Hoo boy!

The best idea is to build your own. Be sure to include a fan.

The bottom line is to find out what you *really* need to get your printer doing useful things and then pick up as much of that material as quickly and cheaply as you possibly can.

## Obtaining Supplies at Good Prices

Next on our rule list is . . .



All the munchables that get gobbled up by your word processing system will eat you out of house and home if you let them. The prices can force you to use second rate materials and supplies, so be sure to find reasonable and sane sources of paper, ribbons, disks, mailers, formed checks, stationery, labels, and whatever. Making the effort to find such sources saves you bunches of money and gives you a far better product out the door.

On the facing page is a list of some of the supply sources that I use . . .

These sources are the ones that I happen to be using at this writing. Naturally, the instant I find some place that is better or cheaper, it will get my business.

Two comments: Quill has far and away the best pricing on paper, ribbons, and metal daisywheels anywhere, but be sure to wait for Quill's monthly loss leader sales. What you need will usually come back around again in a few months. Secondly, Calumet calls its reuseable disk mailers a *#1 Stay-Flat Mailer*. Pricing is—are you ready for this?—under a dime each in quantity.

**Word Processor Supply Sources  
That I Use**

New ribbons and paper:	QUILL 100 S. Schleiter Road Lincolnshire IL 60069 (312) 634-4800
Ribbon rewinding:	TORRES RIBBONS 416 East State St. Redlands CA 92373 (714) 792-0831
Diablo parts:	THE PRINTER WORKS 1961 Alpine Way Hayward CA 94545 (415) 887-6116
Printer bargains:	COMPUTER SHOPPER Box F Titusville FL 32796 (305) 269-3211
Bulk 3M disks:	ALF PRODUCTS 1315F Nelson Street Denver CO, 80215 (303) 234-0871
Disk mailers:	CALUMET CARTON 16920 State Street S. Holland IL, 60643 (312) 333-6521
Checks and labels:	NEBS 12 South Street Townsend MA 01469 (800) 225-9550

Always ask around for the best pricing and delivery on your word processing needs. As guidelines, medium quality disks should cost you no more than \$1.50, Diablo film ribbons no more than \$2.75, disk mailers not more than a dime, and metal daisywheels should not exceed \$30. Some of the large, no-frills discount "club" stores (Price Club, for instance) are starting to offer some limited word-processing supplies at incredibly low prices. Be sure to check out any in your area.

## Communicating

And for our final heavy rule . . .

---

### **Print Quality Rule Number Four**

---

Let the word processing program and the printer communicate with each other as intelligently as possible.

---

If you do not tell your word processor what printer is hung on its output, your software will assume the printer is something worse than a Model 28 Teletype, and you will get print quality that is both atrocious and slow.

On the other hand, if you let your word processor and printer talk to each other at the highest possible level, you are going to get the best printing quality your system is capable of. The trick is to let the printer do what it does best and let Applewriter do what it does best. For instance, if you have a printer with full microjustification firmware, have your printer, rather than Applewriter, do the justification for you.

Two levels of communication are involved in high-quality printing . . .

- 
1. *Low Level Communication*  
Letting Applewriter send stuff to the printer as quickly and as free of errors as possible.
  2. *High Level Communication*  
Imbedding in the text those commands that activate special printer features when and as they are needed.
- 

Low level communication is nothing more than making sure your interface works. These days most newer printers will simply plug and go with most newer serial or parallel interface setups.

If things do not work well the first time, an imbedded command or the flip of a switch should cure things. The comments that follow mostly apply to attempts at using older printers with oddball parallel or serial cards on a IIe. My rule is "If it ain't broke, don't fix it."

Many, if not most, of the better quality printers communicate serially under standard RS232. The trend is to make everything serial because it

eases the FCC interference hassles on new hardware designs. Both the Apple IIc and the Macintosh offer only serial printing. Third party add-ons are available to go parallel if you have to.

Should you need extra RS232 cables to extend a printer, you can build them by buying press on connectors and flat cables out of most any electronic hardware catalog. Cost is far less than ready made cables, and the result is compact and flexible. Suitable cables may show up also as surplus bargains.

## ***Requirements for a Good Serial Interface***

Needless to say, both ends must use the same baud rate. Hidden switches can be flipped or software commands can be sent to adjust the baud rates. You should always try to set up your computer and printer to communicate at the fastest possible baud rate, preferably at 9600 bits per second. Otherwise, time is wasted passing characters back and forth, and your printer will have to wait every now and then for new characters. That waiting period becomes crucial on daisywheel graphic dumps.

You also should defeat any Apple video display echo. Besides putting unreadable garbage on the screen, the screen scrolling times will further slow down any exchange of characters between your word processor and printer. The old ROM version of the IIe has an especially slow screen scroll.

Most importantly, you should be sure you have handshaking between your printer and your word processor. If the printer ever gets behind, it must have some way of telling the computer to stop sending characters for a while. The usual microcomputer way of handling this is with a busy signal line that goes from printer to computer. The busy signal holds up any characters being sent until they can be used. The object is to have the printer limit its own maximum speed rather than slowing printing just to avoid handshaking.

## ***Summing up . . .***

### **For a Good Serial Interface**

1. Run at the fastest baud rate.
2. Defeat the Apple screen echo.
3. Provide busy handshaking.

## The RS232 Interface

As you have probably discovered by now, *RS232 Compatible* means only that you do not have to call the fire department before you plug two RS232 interface connectors into each other. The compatibility in *no* way guarantees that the two devices will actually talk to each other or do what you expect of them. The number and use of each wire in an RS232 interface differs somewhat with each printer application. Figure 3.1 summarizes the important RS232 signal lines.

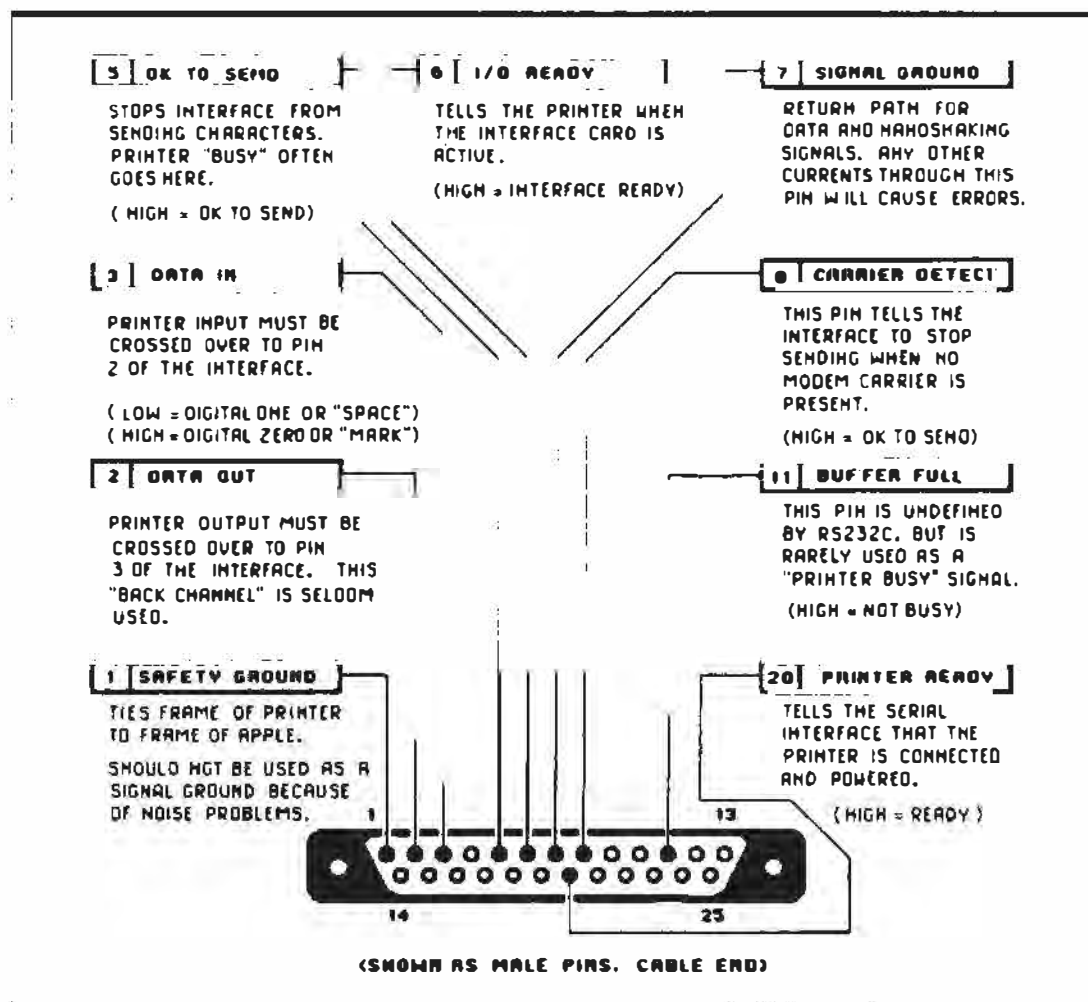
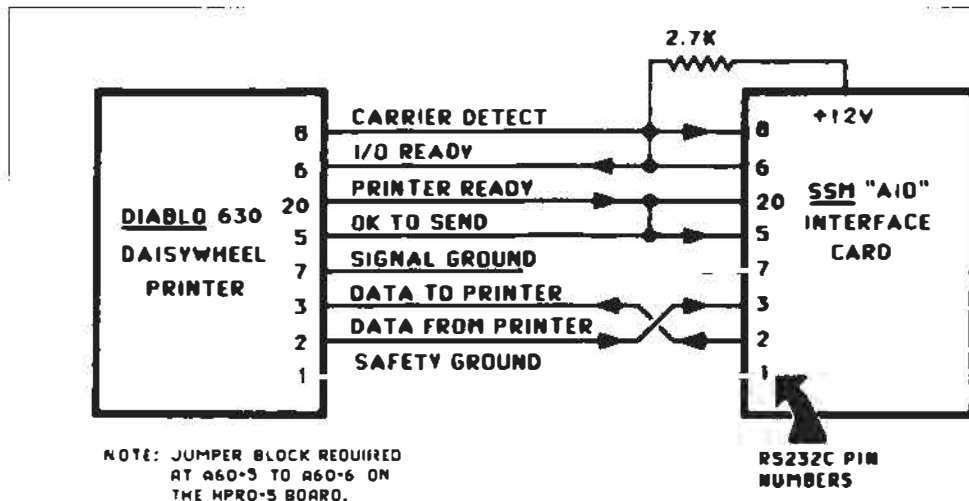


Fig. 3.1. RS232C pin connections normally used by a serial printer interface.

Figure 3.2 shows the custom interface that I use between an Apple IIe with a Mountain Hardware AIO card in slot one. This older interface lashup is more or less typical, but details may change with your needs.





**Fig. 3.2.** Some customizing is needed to get an older RS232C printer interface to work properly. This example links a SSM "P10" interface card to a Diablo 630 printer.

As noted before, if you use a popular printer with a popular interface, you can just plug and go without many problems. If you try an oddball printer or a obscure I/O printer card, some playing around will be needed to get everything flying right side up. Such customizing usually involves setting some DIP switches, crossing a pair of wires, and jumpering some other wires or lines together. Some intelligent printer cards, such as the Grappler, can really foul up the ProDOS Applewriter 2.0 output. See Chapter 6 and Appendix B for the needed patches.

Returning to RS232, note that the data out and data in lines, pins two and three, must be crossed once and only once. The reason is that what is output from the Apple is input to the printer and vice versa. Commercial "modem eliminators" are available that will cross these two wires for you.

Pricing is unreal.

In many RS232 interfaces, you can simply tie pins 6, 8, and 20 together. Pin 20 is usually the busy signal line from printer to computer. Unfortunately, this introduces a bug in the extended Diablo 630 that swallows two characters out of each buffer loading, so pins six and eight are separately held active as shown in Figure 3.2, which is done by using a pullup resistor.

Incidentally, the circuit of Figure 3.2 only works on a Diablo 630 that has the little blue jumper placed between pins A60-5 and A60-6 on the HPRO5 board. Of course, the fun starts when nobody bothers to tell you about little blue jumpers. Details like this explain why the special printer manuals are so important.

If you are having interface problems, first make sure your baud rates are the same on both ends, as are such things as the word length, number of stop bits, forced carriage returns, and the type of parity in use. Then make sure your interface has pins two and three crossed in both directions. As an

RS232 baseline, cross three to two, two to three and separately jumper pins six, eight, and 20 together. Next, try printing at the lowest possible baud rate, preferably 110 baud. This will separate any fundamental gotchas from handshaking problems.

Finally, go up to full speed to resolve any handshaking problems. Handshaking problems usually do not show up until after a few hundred characters have been properly printed. If all the preceding fails, use an oscilloscope or a voltmeter to check out which lines are doing what to whom.

The Apple IIc greatly eases most interface hassles. See the *IIc Technical Reference Manual* for full details on how the serial ports work.

You also can get configuration software that will adjust the ports for you. You can use imbedded [I] commands to set line width, baud rate, etc. This IIc firmware even has an [I]-Z that stops most card interference with commands passed to your printer.

Some of the "toy" daisywheels will accept only low ASCII control commands. If your underlining will not work and gives you a character, an underline, a character, an underline, etc. check this detail. Sometimes you can get underlining by sending seven bit words out your interface. Other times, you can flip a couple of hidden switches or do some minor surgery.

Much more information on interface fundamentals appear in my *TV Typewriter Cookbook*, *Micro Cookbook*, Volume I, and *Micro Cookbook*, Volume II, (SAMS #21313, 21828, and 21829).

All of which summarizes the four most important print quality rules and gets you some interface guidelines. These ought to work for you, no matter where you are or what you have now in the way of hard copy.

## ***For Still More Print Quality***

If you are willing to spend more to get more, the following are some more ways to upgrade print quality.

### **For Best Print Quality**

1. Use a daisywheel rather than a dot matrix printer.
2. Use a real rather than a toy daisywheel printer.
3. Use metal rather than plastic daisywheel elements.
4. Use proportionally spaced rather than fixed pitch wheels.
5. Use a printwheel font with a typestyle that matches the image that you want to project.

6. Use film rather than fabric ribbons.
7. Use fresh ribbons from a quality source.
8. Use a bidirectional rather than a unidirectional print tractor.
9. Use the slowest possible printing speed with maximum settling time.
10. Use the highest quality paper that suits the job to be done.
11. Use a ribbon and paper combination that work well together.
12. Use microjustification rather than full space justification.
13. Use your maintenance manual to keep the printer fine tuned.
14. Use a new platen rather than one that is two years old.
15. Use every feature you can, as long as that feature improves your final product.

## ***Daisywheel versus Dot Matrix Printers***

Daisywheel print quality is vastly better than dot matrix print quality. Period.

Although even the toy daisywheels will do a reasonable printing job, only the real daisywheels give you top drawer quality.

At this writing, only three mainstream real daisywheels exist. These are the Qume® Sprint® series, the Diablo 630, and the heavier NEC® Spinwriters®. Technically, the NEC is really a thimble printer rather than a daisywheel, but you end up with essentially the same results and essentially the same print quality.

As much difference in print quality exists between a metal daisywheel element and a plastic one as between daisywheel and dot matrix print quality. Admittedly the metal wheels have a much higher first cost and are very easily damaged, but for superb results, there is no contest. Note that the print elements are thicker on a metal daisywheel than those on plastic daisywheels. Thus the optimum printer hammer setting for metal is unsuitable for plastic and vice versa. Stick with all metal daisywheels rather than continually readjusting your machine. Otherwise, you whap the metal too hard and the plastic too light.

The best daisywheel elements will offer proportional spacing rather than fixed spacing, which means that thin characters are printed close together and fat characters are printed far apart. With proportional spacing, a capital *W* takes up more room than a lowercase *i*. This spacing is just like real printing but unlike your usual typewriter.

Proportionally spaced printing is far more readable. You can often cram more message in less space as well.

You will find hundreds of different daisywheels available. The type font you pick determines the overall effect of your message and the tone with which it is to be received. Experiment to get the best results. I personally like the BOLD PS wheel for people style communications and the TITAN 10 for machine language dumps and other computer listings. As a reminder, we found out how to automatically handle oddball spokes on offbeat wheels in the previous chapter by using the WPL.SPOKE REARRANGER.

## ***Film versus Cloth Ribbons***

There's absolutely no comparison between film and cloth ribbons.

Film is sharper, blacker, and far better looking. High quality fresh ribbons from a source you trust will give you better and more uniform results. Often though, the house brands will be just as good and far cheaper than genuine name-brand stock. Checking pays. Never nurse a sick ribbon. Flush it as soon as it even threatens trouble.

Color ribbons often end up as bad news. They forever seem to be jamming and needing repairs. Besides color ribbons just don't produce copy that looks that great. Hopefully, these will improve.

Remember that dot matrix printers *must* use cloth ribbons. Film is a no-no.

## ***Print Tractors***

For best quality, you will need a two-way, or *bidirectional* print tractor. The bidirectional tractor positions the paper far more precisely. It is essential for clean graphics or for printing pages that contain several text columns. A bidirectional tractor can fairly simply back up as much as a full page. Any reverse motion at all gets sticky quickly with one-way tractors.

Just as some dot matrix printers will give you higher quality by slowing down and printing more dots per character, some daisywheels will let you slow down your printing by increasing the carriage settling time. Increased settling time gives you more accurate character hits, which are essential for shadow printing and may improve other results. It is often a good idea to run your final out-the-door copy as *slowly* as possible.

## ***Camera-Ready Printing***

Some of the newest daisywheels offer a camera-ready print mode. Usually, these printers just shift the ribbon into high gear so that each character gets a fresh chunk of ribbon for its own private use. Normally,

each hit causes only a 1/5-character ribbon advance, so the price for camera-ready quality is higher ribbon cost. You may find this a good tradeoff but worthwhile only for your final copy.

You'll find a WPL camera-ready printing program in Program A.9. This program gives superb quality automatically on most any daisywheel printer and at only a double cost ribbon penalty.

## ***Microjustification***

*Microjustification* means that you fill out a line by uniformly expanding each space and, if needed, the space between letters. The adjustment can be as small as 1/120th of an inch, which is a tiny fraction of the width of a full character. The fill justify option on Applewriter only lets you do whole space justification, which creates an awkward shading across the text and is visually jarring. Microjustification also can be used to adjust side-by-side characters individually so that they look as natural together as possible. We will look at some fully automatic microjustification and proportional spacing programs in Chapter 4.

## ***Printer Adjustment***

Keeping your printer properly adjusted is very important. The quality of the printing produced by just about any machine will deteriorate with time. Important things to do every few months on a daisywheel printer include washing the printwheels and platen with suitable solvents, adjusting the print hammer mechanism, freeing up the tractor, correcting any linefeed stepper backlash, and doing a general cleaning and lubrication. Once again, you'll find the maintenance manuals essential to help you do the job right. Special tools and gauges may also be needed.

## ***Printer Paper***

Your choice of printer paper should be obvious. At the very least, use 20-pound, extra-white, microperf paper for anything except rough drafts. This stuff runs less on sale than a penny a sheet.

Where customer acceptance is critical, step up to classic laid papers or bond papers with a high rag content.

If you are going to have your final copy offset printed, use a super white, slick surface, hard coated stock for your camera-ready copy. The litho camera used to make printing plates will give you much sharper characters from this type of stock. Your local printer can recommend sources. You'll find Hewlett-Packard plotter paper makes a fairly cheap but workable substitute. Some ribbons work well with certain papers and poorly with others, so test carefully, then match your paper and ribbon to each other.

## ***Printer Platens***

Replacing the platen at least every two years is a good idea. Replace it whether it needs to be replaced or not. Naturally, if the platen looks bad at any time, replace it promptly. Platens on personal computer printers tend to wear unevenly because much of the copy consists of narrow listings or machine dumps. Sometimes a mildly worn platen can be flipped end-for-end to extend its life.

## ***Bells and Whistles***

Finally, nothing is worse than failing to use an existing printer feature that genuinely will help your hard copy. Be sure to learn each bell and whistle on *both* your printer and word processor. Then figure out how to combine them in new, original, and useful ways.

## ***That Old WD40 Ploy***

If you are using a dot matrix printer, you are stuck with cloth ribbons because film ribbons cannot withstand the impact of the printhead pins. What can you do to get the best cloth ribbon life and the best possible impressions? First, keep one almost new ribbon and one reasonably legible ribbon on hand. Use the newer ribbon only when you are doing a final draft of something.

Secondly is the old WD40 ploy. That it works at all is what's astounding. The amazing property of WD40 is that a small amount of it will cause the remaining ink on the ribbon to move laterally by capillary action from the unused portion of the cloth into the empty tracks. You can recycle a ribbon a dozen times or more, as long as the cloth is not physically damaged. How much blacker an old ribbon will become is utterly amazing. Getting just enough WD40 is tricky. Too much is worse than none at all. Here's how you go about . . .

### **Recycling a Dot Matrix Cloth Ribbon**

1. Take the ribbon *far* away from your computer area, then pry the lid off with a pocketknife.
2. Lightly spray the ribbon with WD40. Thumb the maze and lightly spray again.
3. Crank the ribbon once around and lightly spray again.

4. Wipe all the WD40 off everything except the cloth itself. Let the ribbon sit overnight before snapping the cover back on.
5. Test the quality of the ribbon before using it for anything fancy.

If you use too much WD40, the ink will run on the page and look awful. You can also get some bleedthrough and grease spots on the back of your printout. With just the right amount of WD40, you will find this ploy saves you bunches on ribbon costs, and you can use these recycled ribbons over and over again for all but your final draft needs.

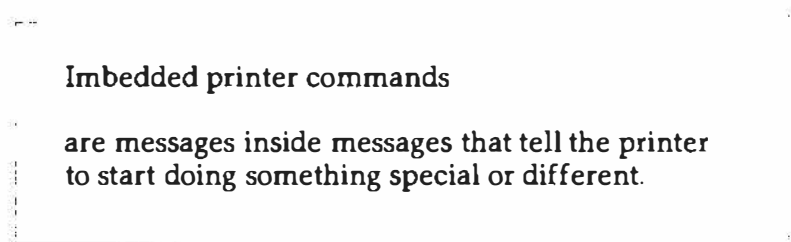
A related stunt is called the *mobius* trick. If your ribbon only has one worn and off-center track, open the case, give the ribbon a half turn, then crank it through. You may have to manually help the half-turn get by internal ribbon guides. After cranking all the way through, your ribbon is now inside out, and you should have a fresh track ready for use.

Naturally, this technique will not work on a ribbon with a centered worn track. Some newer ribbon manufacturers already wind their ribbons as mobius bands, giving each ribbon twice the life to start with. You can tell these ribbons by their double worn track after they have been used.

## Imbedding Print Commands

There's several ways to pass high-level commands back and forth between a word processing computer and a printer. Some of these "older" methods involve hardware switch flipping, some involve configuration jumpers, and others involve direct pokes or stores to certain memory locations.

Tbdy, most printer intelligence is passed back and forth with easily used . . .



Remember that we are using the WPL method of showing control keys here: [esc] means *press the escape key*. [L] means *press the control key, press and release the L key, then release the control key*.

Let's look at three examples of imbedded commands. The Epson command `[esc]-4` tells the printer to begin printing in italics, and `[esc]-5` turns italics back off. Similar commands will set the boldness of the printing, italicize or underline, or specify the number of characters per inch vertically or horizontally.

Turning to daisywheels, the enhanced Diablo command of `[esc]-M` will turn on the microjustify feature. Use `[esc]-X` to turn microjustify back off again. Similar imbedded commands alter margins, spacing, graphics selections, proportional printing, shadow printing, and the like.

Some interface circuits also will need imbedded commands. More correctly, these commands are *communications* commands rather than *printer* commands. One familiar example is the `[I] 80N` command used by parallel interface cards to set the line width to 80 characters and cancel the Apple video echo. The ports on the Apple IIc also accept `[I]` commands to set their line width and video echo features.

See the *IIc Technical Reference Manual* for more details.

Because you normally want Applewriter to set your actual right margin, you should set the interface card's right margin as high as possible. Better yet, defeat any forced carriage returns entirely. Subtle page-creep problems can result from an interface card that is forcing its own carriage returns. If you are going to use all of the really neat features of your particular printer and interface, you have to be able to understand and use imbedded print commands. More importantly, you have to figure out how to let Applewriter handle these commands for you.

Summarizing . . .

#### To Understand Imbedded Commands

1. Find a list of imbedded commands.
2. Play with them one on one.
3. Add them to Applewriter.

I like to do things by hand and by myself when exploring something new. Apply this idea to imbedding your own commands. Get a list of special commands for your printer, then hand list them. Rearrange things in order of likely interest to you.

After that, spend some time "exercising" each new feature until you thoroughly understand what each command can do for you. Be sure to check each of these commands well ahead of time as a separate study instead of trying to figure out something in the middle of some real printing. Be sure to explore both what the imbedded command is intended to do as well as what it really can do for you if you use it in new and unexplored ways.



How do you imbed commands? With Applewriter, you can imbed print commands at three different levels.

### Imbedding Commands in Applewriter

1. Verbatim Method  
Use the [V] command to plant control characters in your text when and as they are needed.
2. Glossary Method  
Use the glossary to give single keystroke entries of long imbedded commands.
3. WPL Method  
Use the WPL word-processing language to enter or strip whole documents of needed commands.

The verbatim method is the simplest. Use it to immediately put an occasional control character directly into your text. Use this method for practice and for seldom used or oddball commands.

The glossary method lets you shorten each often used series of imbedded commands into a single keystroke. This saves looking up complicated commands that you repeatedly use. The glossary selections are put into your text file in the first place by using the verbatim method. The glossary is then saved to disk for later use. You can self prompt a glossary or even run a WPL module off disk with a single keystroke. We will see how to do this in Chapter 5.

The WPL method is a super heavy.

Under automatic program control, WPL can access a document spread over as many drives as you have in your system. WPL will then scan the document and automatically insert or remove any and all imbedded commands of your choice, anyway you want. One big plus of the WPL method is that you never have to look at imbedded commands or work directly with text that has commands imbedded in it.

You take a plain old text file, done the way you already know, then use WPL to automatically imbed the commands immediately *before* each printing. Changes or corrections are always made to the original, pre-imbedded document.

The WPL method is particularly handy if you have to send your files to, say, both a daisywheel printer and a phototypesetter. Each will have its own set of wildly different imbedded commands. Your text files are better with a minimum of imbedding. Customize the files as needed for each output.

We will be using WPL later for invisible and automatic micro-justification and proportional spacing. Here's a quick look at each imbedding method . . .

## Verbatim Method

The [V] key tends to drive new Applewriter users up the wall.

If you accidentally hit [V], you seem to lose control, and all sorts of ungood things start happening to your text. An inverse *H* is added every time you try to backspace, and every attempted deletion *adds* a new character. What's going on here? Remember that we use the [V] symbol to mean *hold down the control key, press and release the capital V key, then release the control key*.

When you press [V], you tell Applewriter *until further notice, I want you to ignore all the control commands that I give you except for [V] and [M]. Instead, you are to directly imbed any other control commands in your text file*. Applewriter then ignores all control commands except for [V] and [M]. Thus, after an accidental [V] press, the left arrow is ignored. Instead, a backspace is imbedded in your text as an [H], tabs become [I], and so on. The more you stir this mess, the worse it gets. The *only* commands the program will recognize are [V] and the carriage return [M].

Unfortunately there are several different ways to show control commands. Figure 3.3 lists the 33 ASCII control commands, their traditional names, and how they are keyed from your Apple. For instance, we see that ASCII code hex \$11 or decimal 17 is called *DC1*, short for *device control one* and is entered from your Apple keyboard by a [Q].

Note that any and all ASCII control commands can be entered directly from the IIe keyboard. Some, such as [Q], require that you hold down the control key. Others, such as [esc] or [tab] will *directly* generate the needed code, without needing the control key pressed.

While we are looking at charts, Figure 3.4 shows some N and N-1 values. Many intelligent printers "expect" to receive certain numeric values in the form of an equivalent ASCII character. This expected character is roughly akin to the BASIC language CHR\$ command that sends out a return, CR, for a CHR\$(13) and so on.

The reason for this oddball turn of events is that a numeric value can be sent as a single ASCII byte rather than as two or three. No guessing is needed to determine whether a received *1* is really a *1*, the start of a *12*, or the beginning of a *123* sequence. Thus, instead of a *101*, you send a lowercase *f* instead. One fixed byte instead of three variable ones.

Sometimes, the actual or N value is used. Other times, an N-1 value is sent. Whether N or N-1 is used depends on the command and the printer. Line and tab values are often sent with N commands, and VMI and HMI vertical and horizontal motions are often handled by the N-1 values. See the advanced manuals on your particular printer for full details.

We are not going to study the individual commands of individual printers in depth. First, because you *must* do this on your own if you are to get the most bang for the buck from your printer. Secondly, because so many different printers are out there. And lastly, because you don't have to worry about such things if you use the invisible and automatic methods that I am about to describe.

ASCII	Hex Code	Dec Code	Ile Keys	Original Use
.....				
NUL	\$00	00	[@] *	Do nothing or null
SOH	\$01	01	[A]	Start of heading
STX	\$02	02	[B]	Start of text
ETX	\$03	03	[C]	End of text
EOT	\$04	04	[D]	End of transmisson
ENQ	\$05	05	[E]	Enquiry
ACK	\$06	06	[F]	Acknowledge
BEL	\$07	07	[G]	Bell or alarm
BS	\$08	08	[H]	Backspace
HT	\$09	09	[I]	Horixontal tab
LF	\$0A	10	[J]	Linefeed
VT	\$0B	11	[K]	Vertical tab
FF	\$0C	12	[L]	Formfeed
CR	\$0D	13	[M]	Carriage return
SO	\$0E	14	[N]	Shift out
SI	\$0F	15	[O]	Shift in
DLE	\$10	16	[P]	Data link escape
DC1	\$11	17	[Q]	Device control #1
DC2	\$12	18	[R]	Device control #2
DC3	\$13	19	[S]	Device control #3
DC4	\$14	20	[T]	Device control #4
NAK	\$15	21	[U]	Negative acknowledge
SYN	\$16	22	[V]	Synchronous idle
ETB	\$17	23	[W]	End block transmit
CAN	\$18	24	[X]	Cancel
EM	\$19	25	[Y]	End of medium
SUB	\$1A	26	[Z]	Substitute
ESC	\$1B	27	[[	Escape
FS	\$1C	28	[{	Form seperator
GS	\$1D	29	[}]	Group seperator
RS	\$1E	30	[^	Range seperator
US	\$1F	31	[_]	User seperator
DEL	\$7F	127	DELETE *	Delete
.....				
Equivalent keys:				
LEFT ARROW = [H] = BS				
TAB = [I] = HT				
DOWN ARROW = [J] = LF				
UP ARROW = [K] = VT				
RETURN = [M] = CR				
RIGHT ARROW = [U] = NAK				
ESCAPE = [[ = ESC				
Gotchas: Many Apple uses set the ASCII most significant bit. To				
set the MSB, add hex \$80 or decimal 128 to above values.				
* - NUL (\$00 or \$80) and DEL (\$7F or \$FF) are reserved				
for internal use by Applewriter Ile. A patch is				
available to restore NUL. See module six.				

Fig. 3.3. ASCII control codes are needed for most intelligent printer interfaces and can appear in many different forms.

"N-1"	"N"	SEND	"N-1"	"N"	SEND	"N-1"	"N"	SEND
0	1	[A]	44	45	-	88	89	y
1	2	[B]	45	46	.	89	90	z
2	3	[C]	46	47	/	90	91	[
3	4	[D]	47	48	0	91	92	\
4	5	[E]	48	49	1	92	93	]
5	6	[F]	49	50	2	93	94	^
6	7	[G]	50	51	3	94	95	~
7	8	[H]	51	52	4	95	96	+
8	9	[I]	52	53	5	96	97	a
9	10	[J]	53	54	6	97	98	b
10	11	[K]	54	55	7	98	99	c
11	12	[L]	55	56	8	99	100	d
12	13	[M]	56	57	9	100	101	e
13	14	[N]	57	58	:	101	102	f
14	15	[O]	58	59	;	102	103	g
15	16	[P]	59	60	<	103	104	h
16	17	[Q]	60	61	=	104	105	i
17	18	[R]	61	62	>	105	106	j
18	19	[S]	62	63	?	106	107	k
19	20	[T]	63	64	@	107	108	l
20	21	[U]	64	65	A	108	109	m
21	22	[V]	65	66	B	109	110	n
22	23	[W]	66	67	C	110	111	o
23	24	[X]	67	68	D	111	112	p
24	25	[Y]	68	69	E	112	113	q
25	26	[Z]	69	70	F	113	114	r
26	27	[{]	70	71	G	114	115	s
27	28	[ ]	71	72	H	115	116	t
28	29	[}]	72	73	I	116	117	u
29	30	[^]	73	74	J	117	118	v
30	31	[_]	74	75	K	118	119	w
31	32	(space)	75	76	L	119	120	x
32	33	!	76	77	M	120	121	y
33	34	"	77	78	N	121	122	z
34	35	#	78	79	O	122	123	(
35	36	\$	79	80	P	123	124	
36	37	@	80	81	Q	124	125	)
37	38	&	81	82	R	125	126	~
38	39	'	82	83	S			
39	40	(	83	84	T			
40	41	)	84	85	U			
41	42	*	85	86	V			
42	43	+	86	87	W			
43	44	,	87	88	X			

.....

"N-1" values are often used by RMI and VMI motion commands.  
 "N" values are often used for lines/page and tab values.

ASCII values of \$00, \$7F, \$80 and \$FF are reserved for internal use by Applewriter IIe and should not be imbedded into textfiles.

**Fig. 3.4.** Many intelligent printers require that command values be passed to them by the numeric N or N-1 value of an ASCII character.

How do you use [V]? Suppose that you want to switch your Epson to print italics, so you want to imbed an *[esc]-4* in your text. To do this using [V], first go to the place in the text where you want to imbed the command. Then type [V][esc][V]4. The first [V] says *verbatim enter the esc key in the text*. The second [V] says *quit imbedding funny commands and switch back to normal word processor use of the control commands*. Note in this example that we need a plain old 4 and not a [4].

Watch this detail very carefully. Provide control characters *only* when they are called for.

As an enhanced Diablo example, the command *[esc] M* turns on microjustification. To imbed this command in your document, go to the right place, then type [V][esc][V]M, and you are home free.

Obviously . . .

---

Do not forget to press the second [V] when you finish imbedding a command!

---

Let's look at another simple example of the verbatim method.

One of the weaknesses of Applewriter's underline mode is that it gives you no direct way to underline up to a period, comma, or question mark. At least not without also underlining the punctuation or adding at least one unwanted space.

This problem has both "micrometer" and "sledgehammer" solutions. The micrometer way is to imbed a backspace *after* the trailing underline token but *before* the comma or period. For instance, you type \zorch \ [V][H][V] to underline the *zorch* but not the period. The backspace swallows the space forced by the underline processor. The imbedded backspace does work on any printer bright enough to recognize a backspace when the printer sees one. The imbedded backspace is a simple fix to the one problem that seems to bother beginning Applewriter users the most.

The sledgehammer solution is to imbed commands that turn on and off the printer's own internal underlining firmware. This is far more flexible. The usual result is a cleaner underline because the printer will properly step up the ribbon advance. Imbedded backspaces are also not needed with this solution.

Better yet, do you really want to underline? How about a doublestrike, a font change, a switch to italics, or a shadow print instead?

Some solutions . . .

### Three Ways To Let Applewriter IIf Underline Up to a Comma or Period

1. Imbed a backspace before the punctuation.
2. Use the underline feature of the printer instead.
3. Substitute shadow printing, bold, or italic fonts.

A subtle gotcha is involved with [V]. If you really want to imbed a [V] in a text file, you have to get sneaky. The single line glossary entry mode will let you directly generate a [V], but the full editing, normal operating mode will not. One simple answer is to put a [V] manually into your glossary and then call [V] into your text over and over as needed. You can also enter a [V] by using [F]ind in its replacement mode.

Two other imbeddings may give you fits.

If you imbed a reverse slash ( \ ) and you are using the same symbol for your internal underlining, you may end up with serious problems. For instance, the Diablo command to disable backward printing uses the reverse slash. Be sure to go out of your way to avoid using reverse slashes for anything but underline commands. Among other reasons, this symbol is not available on all printers. The magic sequence of { < also must be avoided because this sequence turns on the footnote "machine."

So how did I just print it?

## Imbedding with the Glossary

The intermediate way to imbed print commands with Applewriter IIf is to use the glossary. This way a single and meaningful keystroke imbeds the whole command.

No muss, no fuss, no bother.

For instance, you can use [open apple]-I to start italics and [open apple]-i to turn them back off. Big I turns on italics. Little i turns off the italics.

Or, on an enhanced Diablo, an [open apple]-J starts microjustification and [open apple]-j stops microjustification. Big J turns on microjustification. Little j turns off microjustification. We will see some self-prompting glossaries for the Apple DMP, Diablo, Epson, and Imagewriter printers in Chapter 5.

Few people realize that glossary entries can be used to directly execute WPL and other machine commands in Applewriter, besides the usual entering of text in your main text file. In Applewriter, any control characters imbedded in a glossary perform their usual editing functions. Any control characters preceded and followed by a [V] and imbedded in a

glossary are actually imbedded in your text. Remember that a carriage return is substituted automatically for each ] in the glossary.

Note that the NULLIFIER program of Program A.1 may be needed for AWIIe if certain features are to be made available on certain printers. In particular, the Epson underline and superscript need NULLs in your text files. On ProDOS Applewriter 2.0, the user separator [ ] is used as a substitute NULL. Note also that the STRETCHIFIER program of Program A.2 is needed if you are using imbedded control commands. Otherwise, you get the shortline problem in which imbedded commands are counted as real characters.

## A Glossary Example

We will see several self-prompting glossaries in Chapter 5. For now, let's look at a quick and simple glossary use. Sometimes, you may want to combine several commands at once into a macro for your glossary.

One of the stickiest problems involving proportional spacing is that columns of figures get messed up, as do alignments of addresses or anything else that is supposed to be lined up in the middle of a long string of characters. This includes menus, price lists, and anywhere else you need both proportional printing and letterspaced alignment. The usual way out of this bind is to set tabs that align any needed columns. Most proportionally spaced fonts have constant values for all the numbers, so numbers will align on proportional spacing if they start at the same position. Note that the tabs must be set inside *the printer* and not inside your word processor, for only the printer knows where it happens to be on a proportionally spaced line at any instant.

Figure 3.5 shows a worst-case example of exiting proportional spacing at a random point on a line, putting down an aligned row of dots at fixed pitch, then resuming proportional spacing with an aligned column.

THE DON LANCASTER LIBRARY		
Apple II & IIe Assembly Cookbook .....	SAMS #22331	
Micro Cookbook I (Fundamentals) .....	SAMS #21828	
Micro Cookbook II (Machine Language) .....	SAMS #21829	
Enhancing your Apple II vol I (2nd ed.) .....	SAMS #21846	
Enhancing your Apple II vol II .....	SAMS #22425	
Hexadecimal Chronicles .....	SAMS #21802	
CMOS Cookbook .....	SAMS #21398	
TTL Cookbook .....	SAMS #21035	
TV Typewriter Cookbook .....	SAMS #21313	
Cheap Video Cookbook.....	SAMS #21524	
Son of Cheap Video .....	SAMS #21723	
Active Filter Cookbook .....	SAMS #21168	
Incredible Secret Money Machine .....	SAMS #21562	

Fig. 3.5. Proper column alignment is one of the trickiest kinds of proportional spacing.

If you try column alignment without getting sneaky, you will find the dots microtagged all over the lot, for each exit of a proportionally spaced line can end anywhere with respect to fixed pitch. Thus, on exiting to 12 pitch, you can end in any of 10 possible dot positions, only one of which is properly aligned.

The Diablo 630 solution is two glossary entries, keyed as a comma and a period. You do the comma command first, then your row of dots, then the period glossary command. If you want spaces instead, you use the same idea—put a glossary comma before the string of spaces and a glossary command period after the string.

Like so . . .

### Three Macro Glossary Tricks for the Diablo 630

1. `[esc] Q [esc] 1 [H] [tab]`  
Exits proportional spacing and enters fixed spacing in mid-line.
2. `[esc] P`  
Restarts proportional spacing in mid-line.
3. `[esc] X [esc] U [esc] D [esc] M`  
Puts proportionally spaced text inside a box formed by asterisks and keeps a fixed left margin inside the box.

What happens is shown in Figure 3.5.

After a book title is printed, you cancel proportional spacing. Then you set a tab. The tab sets to a constant value in 12 pitch rather than the microjustified position you ended up in. Don't sweat exactly where this tab sits. This newly set tab takes the end slop out of the line. Next, you back up one character, then tab to your newly set tab location. This eliminates any microtagging and gets you back in line with plain old 12 pitch spacing. Next, you put down your dots at 12 pitch, fixed spacing. Then you switch back to proportional spacing for the rest of the line.

One minor gotcha: When you do this, some of the columns still may not be aligned. This can be caused by lots of wide characters in one name and a few thin ones in another. But, the column alignment will now be off by *whole* 12 pitch character spaces one way or the other. Simply do a printout and use page highlighters to paint green all the lines that are shy and paint pink all the lines that are long. Then add or remove a fixed pitch dot or space or two as needed so that the lines all come out even.

Let's review.



The first part of the line is put down in full proportional spacing. A tab is set to take out any microtagging. Move your cursor to that tab. Put down fixed pitch dots. Finally, go back to full proportional spacing. To do this, put down your first character string, a comma glossary entry, a string of dots, a period glossary entry, and your final proportional text. Dump to a printer and add or remove any whole dots as needed.

It's that quick and that easy.

The glossary comma command combines moving forward one space, switching off proportional spacing, setting a tab, backspacing, going to the tab you just set and then doing another space. Not all bad for one keystroke. If you like, you can follow with spaces rather than dots.

All that the *[open apple]*- glossary entry does is get you back to proportional spacing. The *[open apple]*-P command does the same thing.

Oh yes, I almost forgot: If you try doing really weird things to your printer, internal printer bugs are nearly certain to come out of the woodwork. We have already seen that the Diablo 630 has a secondline problem which causes problems with any underlining or funny spoke access on the second line of each paragraph. We saw how the secondline problem is cured by printing of any problem paragraph left-to-right only.

Sometimes, really off-the-wall print commands can be used to get you out of some sticky problem. The third glossary example is typical. To fool the printer into starting a microjustification in mid-line, you shift one halfline up and then one halfline down, which convinces the printer that it is on a new line, and justification starts where you need it to.

What I am leading up to is . . .

Strange bugs are likely to crop up if you try using your printer firmware in obscure or oddball ways. Expect this and be willing to experiment your way to a solution.

Like the 2 x 4 and the mule: Whack it once to get its attention, then whack it a second time to get the job done.

An important reminder . . .

In Applewriter, *[open apple]* does the same thing as *[G]*.

So much for the glossary method of imbedding printer commands. The real heavyweight involves . . .

## *Imbedding with WPL*

WPL is a supervisory language that runs an executive controlling program. This does almost everything a person can do, given a long and detailed enough list of which keys to press in what order. A WPL program is activated by a [P], followed by the do command and the program name. We have already seen several short WPL routines in the previous chapter. We will see more WPL use when we get to microjustification in the next chapter.

You write WPL programs the same way that you would any text file. You test and debug text files the same way that you would any computer program. Full details on what WPL is and how it works appear in the *WPL Programming Manual*.

Owning Applewriter and not using WPL is almost as stupid as buying a Porsche to listen to its AM radio. You automatically exclude yourself from more than 95 percent of the potential of Applewriter if you do not make yourself thoroughly WPL literate.

Please note that WPL is intimately linked to Applewriter. To understate, WPL is not easily moved to another word processor. I know of no other word processor at any price that has available anything which is remotely as flexible or powerful as WPL.

Let's look at a mind-blowing use of WPL . . .

## *Camera-Ready Print Quality*

We'll wrap up this chapter with a WPL routine that will dramatically improve your hard copy, nearly to typeset quality.

Program A.9 is called WPL.CAMERA READY and is intended for Diablo daisywheels. It is easily modified for many different printers. This program scans the document and begins each line with the Bold Print and the Slow Down commands so that each and every character gets whapped twice. This gives you far blacker (or browner or bluer or whatever) images that are much more intense and more uniform. The difference is especially noticeable on classic laid papers or where the text is to be photographed for camera ready printing.

The program then goes back through the document and removes the double whapping imbeddings from each Applewriter command line that starts with a period. If the double whapping before these imbedded commands was not removed, the commands would be printed rather than executed.

Finally, the program finds each *[esc]-X* in your text file and resumes double whapping and slow printing afterwards. This is important if you are doing fancy things in mid-line, such as shadow printing or switching in or out of microjustification. If the program did not do this, the remainder of the line would not get double whapped.

The improvement in print quality comes about several ways. First, the extra settling time gives you more accurate hits, particularly on shadow printed titles. Secondly, you get two ribbon hits rather than one, which is important because the multistrike ribbon only advances one-fifth of a character each hit. Thirdly, the first whap, just like the mule's 2 x 4, primes fancy papers so that they can accept ink. The second whap actually puts the ink down.

You'll get best results on a printer that is hitting a tad on the lean side. If you adjust for a light hit, WPL.CAMERA READY will actually lengthen your printer and printwheel life, while at the same time improving your hard copy images.

Several gotchas are connected with this program. You should normally use WPL.CAMERA READY only on your out-the-door final copy because your film ribbon will last only half as long and the printing is slower than usual.

Another more subtle gotcha is that any carriage returns forced by Applewriter at print time will cause the next line to be printed normally rather than double whapped. Typically, the first line in a paragraph ends up double whapped, and all the rest of the lines are hit only once, which really looks bad.

Awful, even.

One way around this is to force all carriage returns in your document. This is no big deal for a mailing list or a simple business letter but otherwise gets old fast. Even if you force carriage returns, any top or bottom lines or page numbers will not get double whapped.

The secret cure is to use the magic of Applewriter's .pd8 command. First format your document, print it to disk using .pd8 exactly as you want the document to appear, correct for everything but double whapping. Call this file TRANSFER or something similar. Then clear your workspace, load TRANSFER, and run WPL.CAMERA READY. Finally, print TRANSFER with "wide open" print constants of .lm=0, .rm=200, .tm=0, .bm=0, .pm=0, .pi=66, .ut, .tl, and .bl. The result will be a printout with everything camera ready, including every line of all paragraphs and top and bottom headers.

Copies of WPL.CAMERA READY and a bonus program called PRT.WIDE OPEN are included on both companion disks to this volume. A fully automatic print quality improver for the Diablo 630 is named AUTO.PD8.WPL and is included as yet another bonus program on the ProDOS 2.0 companion disk.

A final hassle involves very detailed daisy spokes, particularly the trademark and the copyright symbol. It is best to turn off the double whapping immediately before such characters and turn it back on after them.

You can run WPL.CAMERA READY with a single keystroke out of your glossary. For instance, a glossary definition of *1[P]do WPL.CAMERA READY* will upgrade your print quality automatically on an *[open apple] 1*.



---

# 4

---

## **Microjustification and Proportional Spacing**

A unique run time WPL utility  
that will, with the right printer,  
give you full proportional spacing,  
microjustification,  
improved titles,  
spoke repairs,  
better underlining,  
and do it all invisibly and automatically . . .

---



Print quality from either your Applewriter IIe or ProDOS Applewriter 2.0 word processor can be upgraded dramatically by using a fairly simple WPL run time utility. You run the module immediately before printing, so you do not need to imbed strange or conflicting commands in your disk based text files. This invisibility is particularly important when you are using several different printers, are passing text files over a modem, or are doing phototypesetting. With some care, your print quality upgrading can be fully invisible and fully automatic, even when you use already existing text files.

One such utility is called WPL.FORMAT NOFRILLS D630 and appears as Program A.10. It scans the document, converting everything that was fill justified to full letter-by-letter microjustification in increments of 1/120 of an inch. This program sets up full proportional spacing and corrects the Applewriter underliner by replacing all underlines with a solid underline. Yes, the new underline will neatly underline up to a period or comma and uniformly underlines proportionally spaced text. This cures a pair of nasty Applewriter bugs.

The program fixes other printer bugs, including optionally "rearranging" daisywheel spokes as needed for printwheel compatibility. It also converts all double vertical spaces to a space and a half. The program automatically shadow prints and centers titles, expanding them slightly for the best possible appearance.

Although written and intended for the Diablo 630 printer with WP enhanced firmware, the program is easily customized or modified to do most anything you might ask of it. For all of the bells and whistles to work, you will need a printer that has lots of built in smarts. Particularly, the printer *must* have its own proportional spacing and microjustification available as internal firmware routines.

## ***WPL Custom Formatting***

Deciding what to include in any "automatic" program is a real hassle. In this case, I decided to take half and leave half, including only those

things that just about anyone would like to see. I left out some of the fancier and more specialized things I personally use. This program is designed to run only on the Diablo 630 daisywheel and some of the newer so-called Diablo compatible imitations. You can easily rework this formatter for almost any printer.

The secret to fully automatic microjustification and proportional spacing is to let the printer handle these tasks for you. You let Applewriter do what it does best, and let your printer do what it does best, passing commands from one to the other. At the same time, you are free to repair any bugs or other problems in either Applewriter or the printer firmware.

The code also provides for a follow-up detail program. You can use a detail program to do specific things to a specific document. For instance, I sometimes use a detail file to print the title page of a manuscript and to handle justification inside thought boxes.

It sure would be nice to be able to custom format *any* document you already have written. Unfortunately, we have to have some rules, which must be followed if the formatter is to work properly. These rules appear in Fig. 4.1.

1. The machine must be less than 75% full since your textfile will get longer.
2. No footnotes are allowed, unless you split the WPL formatting program into smaller chunks.
3. An imbedded ".db1" must precede the body of what you want automatically formatted.
4. Imbedded ".db2", ".db3", etc. commands may be added as needed for custom or "detail" work.
5. Each and every center justified line must immediately be preceded by a ".cj". A separate ".cj" is needed for each line of a multi-line centered title.
6. Only a single right margin setting is permitted for the entire document. This limitation can be overcome with follow-up detail work or a fancier formatter.
7. The reverse slash is the only symbol allowed for underline on/off calls. No backspaces are to be imbedded at the end of an underline call.
8. Any paragraph that follows a centered title must be preceded by a blank line followed by a ".fj".
9. All imbedded commands must be done in lower case.

**Fig. 4.1.** *These rules must be followed if the automatic formatter is to work properly. Most existing text files can be quickly made to comply.*

You can easily bring older documents up to specification by following the preceding rules. On new text files, follow the rules to begin with. The rules are very easy to live with, and only a few characters need be added to already existing text files. Those characters are added in such places and



such ways that they do not normally interfere with any other use of the text file.

You must not cram the machine to the gills. The text file gets somewhat longer as it is formatted. It is always a good idea to have some daylight left in your machine anyway. This is especially important if you are to tap Applewriter's powerful .pd8 print-to-disk option. More on .pd8 later.

It pays to keep at least 6K of free text file space on a short IIe and at least 10K of free text file space on a IIc or a IIe with extended memory. The way to tell the difference is with the Mem prompt at bootup. The short IIe allows only 27K. The extended memory IIe and the IIc allow 48K. Obviously, if the file is too long, you separate it into a pair of shorter files.

Our next rule says that footnotes are not allowed. The formatter we are about to look at is much longer than WPL's program length limit of 1024 characters when using footnotes. If you must use footnotes, you can rewrite the formatter as a pair of WPL programs. The last line on the first half then runs the second half of the program.

In fact, you can make your formatter program arbitrarily long, running as many pieces as you need to do the job. There's no limit to how fancy an automatic formatter can get. The automatic formatter is activated on a .db1 key. Thus the part of your document where you want formatting to begin must start with a .db1. I like to handle a title page separately, so my first .db1 will go at the start of the body of the text. Should you want to do additional fancy things with your formatting, .db2, .db3, etc. entries can be put where you want them.

A detail program might be needed to handle custom work. Such custom work is not a part of the formatter that we will look at. Instead, the last thing the formatter does is ask for the name of a detail file. If a detail file exists, it is executed. If not, a single carriage return exits you from all formatting.

Another restriction to this formatter is that each center justified line which you want shadow printed and spaced out slightly *must* be *immediately* preceded by a .cj. The .cj must be on the line immediately before the actual text line. Multiple centered lines must have a .cj immediately before each and every line.

The formatter has one underlining restriction. You must use the default reverse slash symbol. No other underlining symbol is allowed, and no reverse slashes can be used anywhere in your original document *except* for underlining start or stop. Even if the text file has no underlining at all, you may not use the reverse slash.

The present rules limit you to a single right margin setting. You are asked for this setting when the formatter is run. To keep the formatter fast and simple, you are asked for this setting in a most user-vicious, ASCII character form. You are given six hints as to the right character. For instance, you are told that a right margin of 70 is a F and 80 is a P. Thus, you can calculate that a right-margin setting of 76 must be coded as a L.

Sorry about that.

This clumsy way is the standard dino way of passing print parameters to a daisywheel printer. A numeric value is passed as the position of the ASCII character in the ASCII code, which is a *reverse CHR\$( )* sort of thing. See the N values of Figure 3.4 for a complete listing.

If you need more than one right margin setting, you can handle each and every change in a separate detail file, keying on .db2, .db3, and so on. Finally, all of your imbedded commands must be typed in lowercase. This restriction greatly simplifies and shortens the formatting program.

If you don't care for any of this, rearrange the scenery to suit yourself. What I am showing you is what I like for me. The beauty and power of WPL is that it can do nearly anything for anybody.

Program A.10 is a fully invisible and automatic formatter that links Applewriter to a Diablo 630 printer. This program adjusts the squashticity, improves the underlining, fixes underlining bugs, rearranges spokes for the BOLD PS printwheel, tightens the vertical spacing, microjustifies the body text, selects proportional spacing with the proper printwheel, shadow prints and spreads out centered titles, adjusts paragraph ends, then calls for a separate detail work file if needed.

Wow.

WPL programs look like a cross between assembler and Pascal. Longer programs such as this one are best shown in compressed form without any extra spaces used for the usual pretty printing. We will see more on WPL in Chapter 7.

A WPL program is an ordinary text file, written as you would write any normal text document. The program is activated on a do command. In this case, immediately before printing, you call for a *[P] do WPL.FORMAT NOFRILLS D630* if you are using AWIIe or a *[P] do FMT.D630.WPL* if you are using ProDOS Applewriter 2.0. The program asks for the right margin setting and does all its neat stuff. At the end, you are asked for your custom detail file, which is then executed. Finally you print as you normally would.

The program is invisible in that you seldom have to look at all the special imbedded commands. Nor do they have to be saved to your disk based text files, so these commands aren't likely to haunt you later. The program is automatic in that it does everything by itself with minimum assistance.

A WPL program executes one line at a time. Each line must begin with a space or a label. The first character following the first space or string of spaces is treated as a control command. Thus a command of *[space] ppr [L]* tells WPL to select the print-program option, then print an [L] formfeed, which clears the screen. A command of *d2 b* tells WPL to go to the beginning of the text. A command of *[space]pgod2* tells WPL to go to the line that starts with the label *d2*.

WPL is a fascinating language with many powerful secret techniques. For more info on WPL, check into its excellent manuals. You can also analyze and modify the many WPL routines on the companion disk. As with practically everything in the computer world, the best way to learn WPL is with hours and hours of hands-on practice.

The auto formatting program is modular. Each module starts with a ppr line that identifies what the module does. Every module ends one line before the next module identifier. The modules are performed one at a time in sequential order. Although modules *contain* loops and logical decisions, none of these are *between* modules.

Two warnings: If you rearrange or alter this code, the underline routines must be done first. Otherwise, imbedded commands or margin settings involving a reverse slash can royally foul up the works. Also, if you rearrange spokes, any modules following the spoke module must use the new value of the character rather than the "original" value. Watch these two details very carefully. Let's look at each module in turn . . .

## Setting Up

The program begins by turning off the display, which speeds things up greatly and allows full screen prompting and status messages. The screen is cleared, and a title and row of dots is displayed. You are then prompted for a right margin setting and are given hints as to suitable characters. This margin setting *must* be entered as a single ASCII character. That character is saved as WPL's \$A string variable. If you always have the same right margin setting, \$A may be defined in program, eliminating any user entry.

## Adjusting Squashticity

The formatter is best used on a copy of Applewriter to which a STRETCHIFIER program has been added. The AWIIe STRETCHIFIER program of Program A.2 eliminates the shortline problem that was described in Chapter 1. See Chapter 6 and Patch B.16 for the ProDOS 2.0 version. The squashticity module is needed only if this shortline modification is not made. It is far better to install a STRETCHIFIER and bypass the squashticity module. If you set Applewriter for left justification, practically all the text lines will end up *shorter* than the right margin setting, which is caused by word wraparound. If you ask your printer to do a full microjustification based on the Applewriter right margin, most lines will be stretched also. Because some printers can both stretch and compress lines during microjustification, it might be better to set an average line so that it gets neither stretched nor shortened.

The squashticity module works by picking a squashticity factor. Four characters worth of squashticity is often a good choice. This number is the difference between the margin settings of Applewriter and the printer. The module finds the .db1 key, then shortens the Applewriter right margin by using a .rm-4 or whatever squashticity factor. The margin is corrected at the end of the document. For squashticity to work properly, all internal right margin changes must be relative.

To repeat, squashticity eases the shortline problem by expanding imbedded commands. A much better method is to run directly the STRETCHIFIER program and delete this module entirely. We have left it in so that you can experiment with it. You also can manually adjust the squashticity by assigning different values to Applewriter's right margin and your formatter's right margin. A paragraph's last line should be adjusted so that it is neither lengthened nor shortened. All the rest of the paragraph lines should be as "unstretched" or "unsquashed" as possible.

## ***Improving Applewriter's Underline***

It is no secret that Applewriter's underliner is not one of its better features. The worst problem is that underlining by backspacing single characters and printing an underline simply is not compatible with proportional spacing. The spacing tables get messed up by each backspace and will give you an irregular and ratty line. The underline will also have gaps and doublestrikes. A second Applewriter underlining problem is its awkwardness in underlining up to a comma or period without throwing an extra space into the works.

The formatter's underline improving module cures both problems. The module starts by scanning the document beyond the .dbl marker. The first reverse slash that the formatter finds begins underlining. The *second* reverse slash ends underlining. This continues "by twos" through the document. The underline improving module works by underlining the entire phrase at once, giving you correct proportional spacing and a solid and uniform underline line.

Because the underline module alternates on and off commands, all needed reverse slashes must be present, and no extra or unwanted reverse slashes may be included. Failure to eliminate unnecessary reverse slashes will underline the wrong phrases.

The module continues by scanning the document for each space before a period and each space before a comma. Then the module scans for each *underline end after a period* and each *underline end after a comma*. Finally, the module scans for any spaces following a period at the end of a paragraph. These spaces are removed. All of which adjusts underlining so that no periods or commas are underlined. No extra spaces precede a period or comma.

You might want to extend this module so that it also handles rare punctuation, such as question marks or exclamation points. You seldom underline up to these, and hand repair is fast and simple.

## ***Fixing Underline Bugs***

The previous module corrects all underline bugs caused by Applewriter. Unfortunately, the Diablo 630 also has a bug of its own.

The first 630 right-to-left pass after a mode change to fill justification can garble a printed line. This is called the *secondline problem* and it is even. This bug will mess up any underlines or hidden spoke printing on the second line of each printed paragraph. Both the line spacing and content will get fouled up. This bug is infuriatingly subtle and hard to pin down. It is apparently caused by reuse of some variables or improper initialization in the 630 firmware. This module uses a sledgehammer cure.

A paragraph that contains *any* underlining is printed left to right only. This is done by scanning each paragraph backwards from end to beginning while making any needed adjustments. Hidden spoke printing on the second line (such as an @ or a ] on the BOLD PS wheel) is handled sneakily by underlining something *elsewhere* in the problem paragraph. Such use is rare enough that it is normally not worth repairing.

## Rearranging Daisywheel Spokes

As we have seen, not all spokes of all daisywheel elements are directly Applewriter compatible.

The TITAN 10 and MAJESTIC PS wheels are, but the popular BOLD PS wheel is not. The BOLD PS wheel has 15 problem spokes. Of these, seven can be repaired by remapping the characters. The others require special treatment. This module scans the entire document and remaps !, [, ], >, <, |, and @ so that these characters will print properly.

My version of this formatter also substitutes the trademark symbol for a tilde and a copyright symbol for an opening single quote. I have not included these symbols in this listing because you might like to do something else with them. For instance, you might like to approximate the up arrow with a dagger.

Note that machine listings look awful when done with *any* proportionally spaced font. Thus you are better off using fixed pitch TITAN 10 for listings and printouts and reserving BOLD PS for people oriented letters, messages, and manuscripts.

It is important that any modules *following* the spoke remapper must respond to the *remapped* characters, but anything *before* the spoke remapper uses the *original* characters. Thus, if you delete this remapping module, certain others that follow may also have to be changed. The spoke remapper is similar to the WPL.SPOKE REARRANGER program described in Chapter 2 and shown in Program A.5. The remappings that are done were shown in Figure 2.1.

## Tightening Vertical Spacing

This module scans the document and changes each vertical double space to a space and a half, which tightens titles and improves the document's appearance.

Messing with the number of lines gets tricky fast, for Applewriter is busy counting its .pl page length with an internal counter that knows nothing about any special printer commands such as spacing changes, vertical or horizontal line feeds, or vertical or horizontal halfline feeds. You have to be very careful when you change vertical spacing, or you will get page creep that will misalign any pages that follow.

The rule is to "undo what you do," so the printer line counter and the Applewriter line counter end up agreeing at the bottom of each page. To change vertical spacing, the spacing module replaces the *first* two successive carriage returns with one carriage return and one *downward* halfline feed. The *second* pair of two successive carriage returns is replaced with two carriage returns and one *upward* halfline feed. Applewriter will space one, then two, for a total of three. The printer will space one-and-one-half twice, again for a total of three. You could conceivably end up with the next page text positioned half a line above or below where the text really belongs, but the text on the page after that will correct itself automatically. If your page alignment is critical, this section of the module can be hand patched. Chances are that even if this misalignment happened you would not notice it anyhow.

## ***Setting Body Microjustification***

This module searches for the .dbl marker and sets up a full microjustification to the right margin you previously have selected. The left margin is set by the Applewriter .lm setting. Microjustification takes place between the first non-space character on the line and your selected right margin.

Proportional spacing is produced, simply by inserting a proportional printwheel or downloading a proportional font and then switching the printer firmware to handle variable-width characters.

The Diablo 630 isn't too bright at times, so this microjustification module uses the 2-x-4 method to set the right margin twice. The first time gets the firmware's attention. The second time makes sure the margin is reset. The result is a full and true microjustification with each character individually spaced in increments of 1/120th of an inch, which justifies a line to nearly typeset quality.

Note that microjustification can be included only if your printer's firmware has the internal capability of doing a full microjustification. Proportional spacing also requires smarts from your printer.

## ***Fixing Paragraph Ends***

The Diablo microjustification firmware can stretch and compact text rubber band style over an enormous range. Such aggressiveness is needed for very narrow columns, like those used in a newspaper or in certain magazines.

But too much is too much.

One place you do *not* want aggressive justification is in the last line of a paragraph. If this line only goes halfway across the page, you do not want to stretch this line clear to the right margin because the line will then look awful. You will want to turn completely off any justification on the last paragraph line.

This module kills justification on the last paragraph line by searching for a period, an exclamation point, or a question mark, followed by a carriage return. Any of these characters can mark the end of a paragraph. Each line that ends with these marks and a hard return has its justification canceled immediately before the punctuation.

The 2-x-4 method is used once again to get the Diablo to pay attention. Those replacements may seem weird and roundabout, but they work. Note that the `^` or caret in this module is really a BOLD PS exclamation point. Be sure to substitute if you do not rearrange spokes.

I also kill justification on a greater than sign (`>`) followed by a carriage return. This lets you insert comments to an editor or similar notes without stretching them out. Delete this feature if you do not want it.

One big gotcha: Paragraphs that end in spaces will *not* be unjustified!

It is very easy to accidentally insert a space or two following the period ending a paragraph, particularly if you are doing lots of editing and rewriting. These end-of-paragraph spaces *must* be eliminated if this module is to work. To handle this hassle, the earlier underline module eliminates a single space at the end of a paragraph. Which should stomp most of the culprits. For multiple spaces, (1) don't put them in in the first place, (2) eliminate them with repeated `[F] < > < > <a` commands, or (3) add a new custom module to automate the process.

## Shadowing Titles

This module improves titles by switching to shadow printing, slowing the printer down for best registration and then stretching the characters apart slightly for best appearance. Shadowing is done by finding each `.cj`, then fixing the line immediately beyond with the proper commands. Additional stretching can be added by changing the `[B]` in the kerning command to `[C]`, `[D]`, or whatever you feel looks best.

I choose to use the Applewriter centering routine rather than the centering firmware in the printer. Both choices have advantages and limitations. With Applewriter centering, the code is simple and short, yet the lines may not be exactly centered, owing to the proportional spacing. Centering should be pretty good if you have made the shortline modification and your titles do not have lots of uppercase letters. It can be further improved by preceding each `.cj` with a `.pm-5` or whatever you feel is appropriate. Creative use of paragraph margin commands or extra spaces can adjust problem titles.

## *Cleaning Up*

The formatting program ends by asking you for the name of a detail work file. If you need one, it is run at this time. Once again, you use a detail file to make custom alterations on one specific text file. If you have no detail file, a carriage return exits you back to Applewriter.

Several sample WPL.DETAIL bonus programs have been added to the companion disk as examples. The ProDOS version even includes a formatter that does its thing, automatically makes a .pd8 transfer copy, sets up camera-ready copy, and prints. All of which is fully unattended. This AUTO.PD8.WPL module also shows you how to beat the apparent 2K limit on a WPL file. In reality, WPL programs can be as long as you need them.

Execution speed depends on how many bells and whistles you put into your formatter and on the length of your text files. Allow a half minute to format a business letter and as much as four to five minutes for a very long and involved text file.

Note that you can upgrade to camera-ready printing after you format. Just run WPL.FORMAT D630 NOFRILLS first and follow with the WPL.CAMERA READY program described in Chapter 3 and shown in Program A.9.

Because WPL.CAMERA READY cannot recognize Applewriter-forced carriage returns, you usually will want to do a .pd8 print to disk first, providing a what-you-see-is-what-you-get hard copy image on the disk. Then you can run a fancier version of WPL.FORMAT D630 NOFRILLS. Finally, run WPL.CAMERA READY. As we saw a few paragraphs back, this print-file sequence is automated under AUTO.PD8.WPL on the ProDOS companion disk.

I have also added as bonus programs two fancier and specialized formatters to the companion disk: WPL.FORMAT BOLD PS and WPL.FORMAT MAJESTIC, which also are for the Diablo 630. Besides doing what WPL.FORMAT D630 NOFRILLS does, these formatters also handle titles, stretch blurbs, and do internal justification of thought boxes.

As a reminder, these WPL routines are what I did for me.

Anything you do not like about them can be fixed, provided, of course, that your printer has some reasonable intelligence. Some printers may demand a way to imbed NULLs in your Applewriter text files and will need the NULLIFIER patch. The STRETCHIFIER can also be used for the best possible appearance.



---

# 5

---

## **Self Prompting a Glossary**

Glossaries can be made much more powerful  
and much easier to use when you title them,  
self prompt them, or use them  
to execute control or WPL commands.

Here's full details,  
along with use glossaries for  
four popular printer families . . .

---



Few people realize how simple it is to make an Applewriter glossary self titling, self prompting, and directly able to run WPL routines or handle hundreds of other complex control commands using single keystrokes. Yet these easy techniques can dramatically improve your word processing abilities.

Self prompting glossaries may be used either with Applewriter IIe or ProDOS Applewriter 2.0. Titling glossaries is handy when you have several of them on a disk and want to tell which is which. A self prompting glossary will quickly display a tutorial help screen after the user presses a single key. This sure beats memorizing obscure keystrokes or hunting for lost documentation. Single key macro execution of a WPL routine or loading a print constant, tab file, or whatever makes for convenient and error free Applewriter use.

## *What Is in a Glossary?*

The intended use of the Applewriter glossary is to let single keystrokes insert frequently used phrases into your text files. For instance, the usual *Sincerely yours* plus your name, firm name, and so on, can all be entered in your text file with a single keystroke. To enter the phrase, you simply hold down [open apple], press the selected key, and the phrase magically inserts itself. We've already seen how a glossary also can be used to pass individual imbedded commands to an intelligent printer or a printer interface.

Very conveniently, the glossary will treat a ] as a substitute for a carriage return. By substituting ] as needed for each carriage return, you can have multiline glossary entries.

The only *disallowed* glossary keys are the /, ?, and \*. You can have a separate glossary entry for each uppercase character, each lowercase character, each number, each key used with the control key, and all punctuation except the three disallowed symbols. A glossary could provide you with more than a hundred different user-defined keys at any one time. The limit on any one glossary is 2048 characters, which is more than is normally needed.

One mind blowing feature of the Applewriter glossary code is that it can call *itself* up to eight times over. Thus, if you have several alike-but-different-somehow phrases, they often can be compacted by sharing common words. Glossary nesting is done in much the same way that a subroutine shortens code when a program accesses the subroutine several different times.

Glossaries can be created one phrase at a time using [G] or may be written all at once as an ordinary text file. A [F]ind and replace operation may be used to insert glossary [V]erbatim characters when and as needed, and ] may be used as substitute carriage returns. Glossaries are loaded into or stored from a special glossary buffer, using the Applewriter [Q]-E and [Q]-F auxiliary function commands.

## *Immediate WPL Execution from the Glossary*

When you study the Applewriter code in detail, you find out that WPL and the glossary share many common routines. Few people realize that you can execute word processing commands out of the glossary in exactly the same way that you would use a glossary to put a phrase into your text file. Repeating that: With Applewriter, you can run WPL in immediate mode from the glossary!

Figure 5.1 shows some simple and fairly obvious uses of this newfound power. You enter a glossary command by starting with a key letter, followed by what you want to have happen. For instance, a glossary assigned to [open apple] z and containing an entry of zzorch will enter the word *zorch* in your text file every time you press [open apple] and lowercase z.

The c [O] as6,d1] glossary entry says *on a lowercase c, catalog the disk that is in drive 1*. Similarly, the C [O] as6,d2] glossary entry says *on an uppercase C, catalog the disk that is in drive 2*. The [O] in either example selects the DOS commands menu.

As you have seen, using bracket pairs to show control commands is more or less standard in both Applewriter and WPL.

In all the listings in this book, a *pair* of brackets means a control command. A single right bracket by itself *means* itself, usually used as a substitute glossary carriage return. Note that final ] in each command, which adds the ending carriage return needed to carry out the actual catalog operation.

The reason you use these fake carriage returns instead of real ones is that glossary entry terminates on a real carriage return. Thus fake carriage returns are needed to force multiline glossary entries or to separate multiple control commands. You can execute a *single* WPL program with one keystroke. The glossary entry of 1 [P] do WPL.CAMERA READY] says

on a numeric one glossary selection, execute the WPL program named WPL.CAMERA READY. After selection, the WPL routine will do its thing.

To Catalog drive #1 on a lowercase "c" and drive #2 on an uppercase "C":

c[0] = 6,411

C[0]a36,d2]

To execute a WPL program named "WPL.CAMERA READY" on a numeric "1":

1[P]downpl.camera ready]

To insert boilerplate module six on a numeric "6":

6[L]boilerplate 6.0,d2]

To insert file modules 1, 2, and 8, on a numeric "7":

7(L)fmod1)[L]fmod2)(L)fmod8]

To make 64 copies of your file on a "1":

[illegible]

To change to "wide open" print constants on an "Q":

@ [P] ? ] t m 0 ] b m 0 ] l m 0 ] r m 2 5 5 ] p m 0 ] p i 6 6 ] p l 6 6 ] t l ] b l ] ]

To return to some "stock" print constants on a "#":

#[P]?[tm4]bm4]lm7]rm70]pm6]pi66]pl62]t1]b1]]

**To do a fast clear-to-end-of-file on a "!":**

! [F] &lt; ?????????????????????????????????????? &lt; A

Notes: [P] means "<control>-P", etc. Brackets not in pairs are real. The glossary will substitute a carriage return for a "]".

**Fig. 5.1.** Many commands may be executed from the AWII glossary in an "immediate" mode. This can be done by typing the [open apple] key along with a single other chosen key.

Unfortunately, you cannot execute a *series* of WPL programs on a single glossary command. The reason is that the WPL *[P] do* command only *loads* a WPL program and sets a flag. The command does *not* actually run the WPL routine. The WPL program will not begin until *after* the glossary activity is completely finished. If you mistakenly try something like a *[P] do WPL.1] [P] do WPL.2*, the program WPL.1 will be loaded and the WPL activity flag will be set. Then the program WPL.2 will be loaded,

overwriting WPL.1. Finally when the glossary is finished, control is transferred to WPL.2. Because WPL cannot start until the glossary is done, only the second program gets run.

One of my favorite Applewriter tricks also appears in the preceding list. In only a few keystrokes, you can change a single file to 64 identical files, which is most handy for printing disk labels, return addresses, and such. What happens is that the *[L]# load from file* command is used six times over. Each use copies the file to itself, doubling its length. Watch out for memory overflow on this one.

The preceding listing also shows how to insert lengthy boilerplate into text files. In contrast to WPL programs, with glossaries you can use a single glossary keystroke to load as many individual files as you care to. The list also shows you how to change all the print constants with a single keystroke. Resetting your print constants becomes important fast if the printer fouls the works in mid-document.

Many sneaky tricks can be done by using Applewriter's .pd8 print to disk option. Once you have printed to disk, you no longer want to add any additional margin padding because now you are in a true what-you-see-is-what-you-get mode. To print a fully formatted file from disk, you should use a set of wide open print constants. The exact disk image then goes to your printer, modem, or phototypesetter. Single key setting of print constants becomes most handy.

The final example in the preceding list shows how to erase to the end of document with a single exclamation point, !. This strange looking glossary entry says *from here to the end, find lots of somethings and replace them with nothing*.

One gotcha: Be sure your data direction is set to > before using this command. By the way, the glossary code in old Applewriter 2.0 works differently from either of the newer Applewriter IIe or ProDOS 2.0 versions we are dealing with here. Old Applewriter 2.0 is horribly obsolete. Its continued use is inexcusable.

## Glossary Restrictions

How do we tell whether a glossary entry will go directly in your text file, will execute immediately, or will imbed a command for your printer? What restrictions are there on your glossary use? Figure 5.2 gives four simple glossary rules.

1. A control character imbedded in a glossary command will try to do its thing immediately when called.  
  
A control character imbedded in a glossary command that is preceeded and followed by a [V] will be verbatim entered into your textfile.
2. The glossary will substitute a carriage return for a "]". The glossary will treat (x), (y), (z), \$A, \$B, \$C, and \$D as ordinary printable characters.
3. WPL will substitute its numeric variables on (x), (y), or (z). WPL will substitute its string variables on \$A, \$B, \$C, or \$D. WPL will treat a "]" as an ordinary printable character.
4. A single glossary entry can run only a single WPL program. If a glossary entry tries to run multiple WPL programs, only the last WPL program will run.  
  
Glossary activity is always completed before a WPL program begins. All the "[P]do" command does is load a WPL routine and set a flag.

Fig. 5.2. Rules for WPL and glossary compatibility.

If your glossary entry has no control commands, the phrase goes directly into your text file. If your entry has control commands *by themselves*, the glossary entry carries out an immediate action, such as loading, storing, printing, running a WPL routine, or doing anything else you can do with the control commands. If each control command in the glossary is preceded by and followed by the verbatim [V], the control commands will be imbedded directly into your text file.

Repeating, an L glossary entry goes in your text file as an uppercase letter. A [L] entry will try to do an immediate load. A [V][L] [V] will imbed a control-L in your text file. Use the first for phrases, the second for immediate commands, and the third to imbed print time commands.

The glossary will substitute a carriage return for each ] the glossary finds in its entries. Sadly, the glossary will *not* recognize WPL variables. Thus an (x) in a glossary entry prints as you see it instead of substituting the numeric value for the WPL x variable. The same is true for strings \$A-\$D.

Interestingly enough, the glossary can change, set, or define WPL variables. It just cannot substitute them into your text file.

WPL behaves just the opposite. WPL will, of course, substitute numerics on its own (x), (y), and (z) variables and will substitute strings on its \$A, \$B, \$C, and \$D strings. But WPL does not recognize ] as anything but a closing bracket.

Reviewing, the [P]do command does not execute a WPL program. All this command does is load the program and set the WPL activity flag. If you try running several WPL programs with a single glossary key, only the *last* WPL routine will execute, after all the earlier routines have been overwritten.

Our glossary restrictions boil down to this: Use letters for phrases. Use control commands for immediate instructions. Use control commands bracketed by [V]s for imbedding control characters into a text file. Multiple commands on a single keystroke can usually be performed, except for WPL routines. Direct substitution of WPL variables cannot be done. Other multiple commands should be separated by fake ] carriage returns.

## Self Titling and Self Prompting

We can use these immediate execution tricks to build any number of glossaries that are self titled and self prompting. All you have to do is carefully organize your glossary. Figure 5.3 shows details.

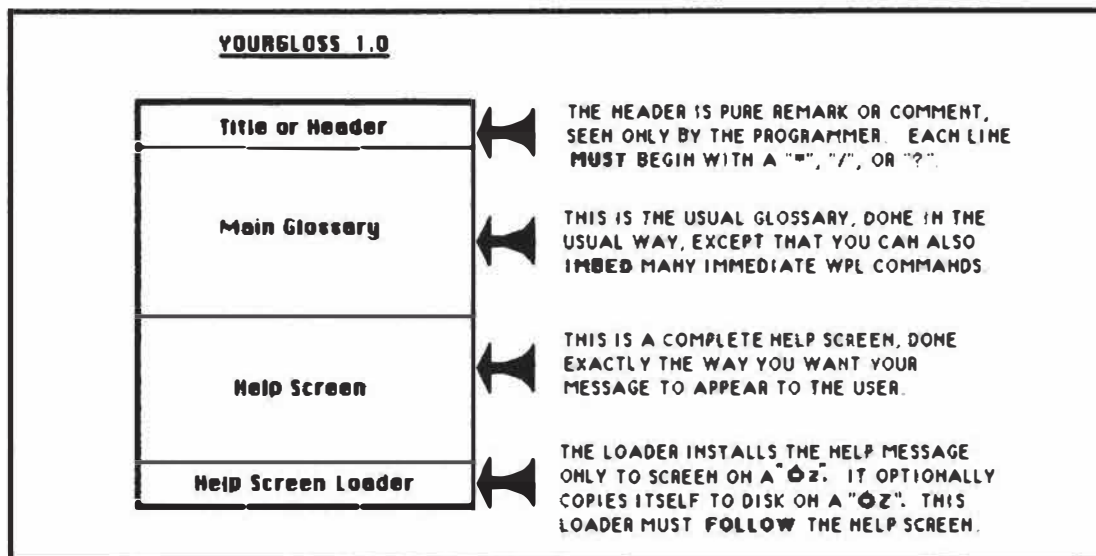


Fig. 5.3. Organizing a self titling and self prompting glossary.

Four parts make up a self titled and self prompting glossary: the *title*, the *main glossary*, the *help screen*, and the *help screen loader*. The title part is trivial. A glossary key of ? or \* gets pre-filtered as a glossary Define or Purge command. Entering [open apple]-/ will get you the main help menu rather than a glossary entry of /. Any glossary line that starts with ?, /, or \* will never be accessed and thus may be used as a title or comment line. Comment lines in a glossary, are seen only by the programmer, and they are very useful to avoid mixups and version hassles.

Figure 5.4 shows how to add comment lines to an AWIIe glossary.

I prefer to use the ?.

The main glossary is the one you do as though you were not going to make it self titling or self prompting. However, we now know we can add all sorts of powerful new features to our glossaries by using immediate commands. You can mix phrases and commands any way you like.



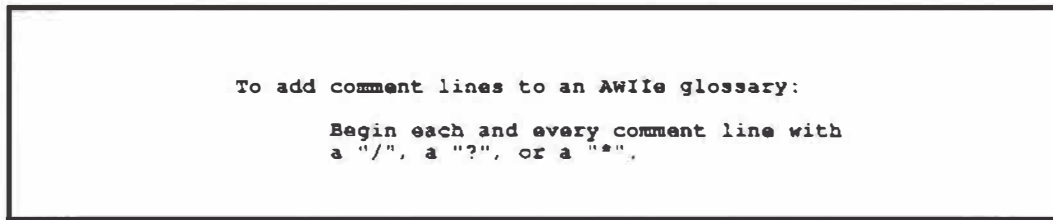


Fig. 5.4. Comment lines in a glossary are seen only by the programmer. They are most useful to avoid mixups and version hassles.

How do you make a glossary self prompting? How do you put up a help screen that, on a single keystroke, summarizes exactly what each glossary key will do? One obvious route is to use glossary commands to load a long help screen directly to your text file.

This has several severe disadvantages. First, a full screen entered from the glossary may take as long as 30 seconds to display. Secondly, you will have altered your text file in the process of putting the help screen in place. Thirdly, when you are finished with the help screen, you somehow have to undo it to get your text files back to the way they were.

Suppose that we get sneaky instead. A glossary is a text file, right? Glossaries also are likely to be on the disk in the active drive. So why not load the entire glossary directly to *screen only*, putting a help message at the end and letting the rest scroll on by?

Better yet, why not just tack a help screen on the end of your glossary? Then load the help screen directly out of the glossary text file, keying on delimiters at the beginning and end of the actual help screen. This method is much faster. In fact it takes all of three seconds. It does not disturb your working text file and exits instantly on a carriage return. No separate help file or filename is needed.

So the third part of your glossary is a help screen that appears exactly as you want your user to see it on the screen. This module may gobble up a thousand characters or so out of your glossary. More often than not, the space is available. If not, you can shorten your help screen as needed.

The fourth and final part of our upgraded glossary is the help screen loader. Note that glossary entries must follow the screen image. Figure 5.5 shows details.

Two loaders are shown, keying on lowercase and uppercase Z.

The lowercase *z* loader tries to load your glossary text file directly to screen, beginning with a *startstring* at the beginning of the screen image and ending with an *endstring* at the end of the screen image. This lowercase *z* loader is the one you use most often and is the faster of the two.

Unfortunately, this loader works only if you have a copy of the current glossary in your active drive. If you just switched disks or drives, you may not have a copy of your glossary in the active drive. This is where the uppercase *Z* loader comes in. On a *Z*, a copy of the current glossary is saved to the currently active drive, and this new glossary image is then loaded to screen only. Use *Z* for a new disk. Use *z* all other times. Loader *z* takes around three seconds and *Z* takes nearly 10.

```

To provide the tutorial to screen only on a lowercase "z":
    z[L]GLOSSNAME<atartstring<endstring<\}

To provide the tutorial to screen and also copy the glossary to
the currently active disk on an uppercase "Z":
    Z[Q]fEGLOSSNAME] [L]EGLOSSNAME<atartstring<endstring<\}

.....

Notes:  {P} means "<control>~P", etc. All single brackets are
        real brackets.

        Use uppercase "Z" the first time you access any new
        diskette. Use lowercase "z" otherwise.

```

**Fig. 5.5.** Self loader commands will quickly load a help tutorial directly to screen. Note that these glossary entries must follow the screen image.

One nasty gotcha: The help screen loader must *follow* the actual help screen. If not, the help screen loader will find the delimiters in *itself* rather than those in the help screen.

Your tutorial help screen also has some minor restrictions. All glossary entries except the loaders must precede the help screen. The help screen itself must be 23 lines or less in length. No screen line is allowed to start with a z or Z as its first character. Any first character on a screen line not previously defined as a real glossary entry could conceivably be entered into your text file through some fumble fingered typing. With most longer glossaries, this screen-line-as-glossary-entry problem never shows up. If the possibility bothers you, put a box of stars around your help screen or start each help screen line with a space. Then predefine *space* as a frequently used glossary entry or as an immediate return.

We can tie things together with these . . .

## Four Examples

What good is all this? Titling a glossary makes it easier on you as a programmer. You now have a sure way to identify each and every version of all your glossaries. Mixups and changes made to the wrong glossary are easily avoided with good titles.

Self prompting a glossary makes it easier on you or your user because a full tutorial help screen can be produced by pressing one key. No more having to look up the keystrokes on a lost sheet of paper. No more having to memorize rarely used glossary commands. No more using dozens of keystrokes when one will do.

Four examples of self titled and self prompting glossaries are shown in Programs A.11 through A.14, all true self prompting glossaries that let you

tap the fancy commands of an intelligent printer, using the smarts of the printer where they can do the most good.

AGLOSS is intended for the Apple letter quality printer. DGLOSS is for the Diablo 630. EGLOSS is for Epson printers. IGLOSS is for the Imagewriter. Yes, an LGLOSS and PGLOSS is in the works for the laser printer. You can write or call for a free copy.

As you probably have guessed by now, I am a Diablo person at this writing, so DGLOSS has been thoroughly tested, wrung out, during a year's heavy use. The only guarantee I offer on AGLOSS, EGLOSS, and IGLOSS is that I tried them and they seem to work. Chances are you will want to modify or improve these glossaries on your own.

If you don't like things the way they are, rearrange them to suit yourself. If a glossary entry does not work on your printer, drag out your manuals and find out why. Changing an entry or two should be a simple matter.

Note how each control character to be imbedded in your text file is bracketed by a pair of [V]s. Note also how the loader follows the help screen. For instance, in EGLOSS, the *startstring* is a carriage return followed by five spaces. The *endstring* is an *ls.* followed by a carriage return. The reverse slash says to load to screen only.

These glossaries work best on a third backup copy of Applewriter that has been patched to allow imbedding NULL commands and to prevent justified shortlines. Details on NULLIFIER and STRETCHIFIER appear elsewhere in this text. The AWIIe NULLIFIER is only needed if your printer requires imbedded NULL commands for certain actions. The ProDOS 2.0 NULLIFIER is needed only if you require *both* NULLs and the stock substitute US user separator [\_\_\_]. Note that older Epson printers needed NULL commands for underline and superscript.

Each glossary action is keyed to an easy-to-remember letter. An uppercase letter is *on*, *up*, *more*, *above*, or *right*. For instance, a DGLOSS S turns on shadow printing and an s turns it off. A J starts microjustification and j stops it. Once again, a z from the glossary gives you a full help screen, and Z gives you a tutorial help screen and saves a new glossary copy to the currently active drive.

The Y and y selections are yours to customize any way you like. In my personal version of DGLOSS, I also use the numeric keys to do things, such as formatting for microjustification and proportional spacing, handling the camera ready double whapping trick, selecting print constants, and much more. I've left these modules out because they are specific to my needs. Details on these modules appear elsewhere in this book and on the companion disk.

Once again, if you don't like the scenery, rearrange it to suit yourself.

Just as with WPL, almost anything goes in a self prompting glossary. All four glossaries are available on both versions of the companion disks. The ProDOS version of DGLOSS also includes automatic prefix setting, automatic formatting, and a few other neat routines.

Other printers will need their own custom glossaries. EGLOSS is a good starting point for most dot matrix printers. DGLOSS is the best baseline for most daisywheels. Note how AGLOSS and DGLOSS are alike-but-different-somehow, as are EGLOSS and IGLOSS. Individual commands and features will, of course, have to be changed after careful study of your printer's manuals and your needs.



---

# 6

---

## Some Patches

Simple machine language mods  
that imbed NULLs, fix short lines,  
improve Ilc status displays,  
open the code for customizing,  
speed up WPL, link new code modules,  
find space on disk,  
restore help menus,  
simplify ProDOS prefixing,  
cure Grappler hassles,  
more . . .

---



Have you wondered how those magic Applesoft repair routines that we showed you back in Chapter 1 work? In Chapter 7, we will tear ProDOS Applewriter 2.0 from stem to stern with a thorough and complete disassembly script and reveal the secrets of capturing your own source code. Before we do though, let's look at those same Applesoft patches done in machine language. This will give you an intermediate level look at the patching process and some insights into what is really happening.

We will look at two groups of machine language patches here.

The first group is only for Applewriter IIe, and the second group is only for ProDOS Applewriter 2.0.

Note several things. First, make patches only to your third or higher backup copy of Applewriter. Do not, under any circumstances, try to patch either of your stock factory disks! Secondly, a patch will only work on one exact version. Should any changes be made in any program code, these patches will not work as intended.

## ***AWIIe Patches***

Seven Applewriter IIe features or repairs seem to be those most asked about on the helpline. As you will see, the patches are much simpler in machine language. On the other hand, they aren't nearly as automatic and there's lots of room for user errors.

First, we will show a way to imbed NULL commands in your text files, so you can handle superscripts and improve underlining on older Epson printers. This patch is similar to the NULLIFIER program described in Chapter 1.

Secondly, a cure is required for the shortline problem that results when imbedded printing commands are counted as real characters in the fill justify mode. This patch is similar to the STRETCHIFIER of Chapter 1.

Thirdly, a way is needed to speed up WPL. Stock WPL can only view the cursed character by way of a very long and elaborate routine. A patch here can put the cursed character directly into a WPL character string. This is most handy for such things as doing automatic tabbing on

assembly listings and other places where decisions need to be made on a character-by-character scanning basis. This patch is called the CURSIFIER. As an example, we will see how to use the CURSIFIER as a numeric scanner to provide a sorely needed AWIIe space-on-disk utility.

Fourthly, a route to open AWIIe for your own custom modifications seems to be much in demand, including the ability to add a PEEK and POKE capability to WPL. This patch is the PATCHIFIER.

Fifthly, a link that lets you or WPL run your own custom machine language module would be most handy for such things as HIRES graphics dumps, plotter commands, and similar exotic extensions. The name for this patch is the LINKIFIER.

Sixthly, a cosmetic glitch trashes the IIc status display if you try using older IIe only versions of the code on a IIc. This patch was described in Chapter 1 as the CLARIFIER Applesoft program.

Our seventh module restores the AWIIe help screens that disappear when you overwrite the otherwise useless Volume Verify module with new and good stuff. This patch is the RESTORIFIER. In fact, the RESTORIFIER is built into the STRETCHIFIER and CLARIFIER code. We have separated this code to be used if you are doing different things on your own.

All these hassles can be cured by making some relatively simple changes to your working code.

Before we begin, though, let's repeat the warning: These patches are in no way approved of or supported by Apple Computer or the original program author. They are simply some useful changes that seem to work for me. Use of these patches is totally at your own risk.

## ***Making an AWIIe Patch***

Listing B.1 shows how to patch an Applewriter IIe program. The versions we will work with here are only for the DOS 3.3e Applewriter IIe program.

Two totally different word processor modules exist on the DOS 3.3 Applewriter IIe disk. At the time you boot the disk, a loader program called OBJ.BOOT is run that tests your machine configuration. If you have a 64K Apple IIe without extended memory, the E code version OBJ.APWRT][E is run for you. If you have extended memory in your expansion slot, which gives you a full 128K of RAM, then the F code version OBJ.APWRT][F is run instead.

The F version of the code is usually the best choice because F code gives you a much larger 48K text file, besides leaving lots of extra room in the machine for your own custom mods.

Once again, you get the E code on booting a 64K machine, and you get the F code on a 128K machine startup. More on Applewriter IIe memory maps appeared in *Enhancing Your Apple II*, Volume II (SAMS #22425). ProDOS Applewriter 2.0 memory maps and a complete disassembly script appear in the next chapter.



The important thing to note about AWIIe is that the E code and the F code are significantly different. The differences are caused by the F version needing extensive memory management of the two 64K memory banks. Although many E and F individual routines are alike-but-different-somehow, the length and the entry points of the E and F code modules all differ. Thus, an E patch will bomb F code and vice versa. Avoiding version mixups is extremely important.

To repeat, work *only* with your *third* or higher backup copy of these programs. Do *not* ever make any patch to either your stock AWIIe disk or its factory supplied backup. Extra backup copies are easily made by any of the usual methods.

I personally use Copy II+ with a parameter change of 10:96.

The program modules OBJ.APWRT][E and OBJ.APWRT][F are standard binary files viewable and modifiable under stock DOS 3.3e. The only little gotcha is that these files are intended to be run starting at location hex \$2300 and must be relocated when they are first loaded in the machine.

To patch either of these program modules, first boot stock DOS 3.3e, then load either module in your Apple IIe, being sure to use an , A\$2300 trailer. Next, you verify the code area where you intend the patch to go. If you do not have an exact match, you must not attempt to complete the patch. If you cannot verify, you have the wrong version, the wrong position in the machine, or have made some other serious error.

For instance, to verify code starting at \$26E1, you would first get to the monitor by doing the usual CALL-151, then you type a 26E1 and a carriage return. Successive carriage returns will get you up to eight additional values per hit. You then compare the hex dump you have just done against the bytes to be verified.

If you do not have an *exact* match, you must not continue.

Once a code area has been verified, you can make the needed patch. To change the code starting at \$26E1, you would type \$26E1, a colon, a space and then as many hex pairs as you need to complete the patch. After your entry is complete, you should once again verify. This verification makes sure that what went in the machine is what you thought you put there.

Typically, a patch will need several areas verified and changed. Some individual mods will be single bytes, but others may need a dozen or more sequential bytes.

When you complete your mods, save the patch to disk under the original filename.

Note particularly that the length of the E code and the F code differ. Be sure to use an E trailer of , A\$2300, L\$2F58 and an F trailer of , A\$2300, L\$30D2. Remember that you can type DOS 3.3e commands directly. You do not have to get back into Applesoft to do so.

You should, of course, test your patches to be sure they are well behaved. Do this testing initially without using any valuable text files. All of the individual pieces of each patch are separately available on the AWIIe companion disk for this volume, and those patches are ready to install.

Let's first look at the individual patches . . .

## ***The NULLIFIER (AWIIe)***

Patch B.1 is called the NULLIFIER and imbeds NULL control commands into your AWIIe text files. Older Applewriter 2.0 imbeds NULL commands by using *[V]@[V]* keystrokes. This feature was dropped when AWIIe was made. Reserved file marker characters in AWIIe are \$00 and \$FF marking the open and closed ends of both the HIFILE and LOFILE internal work areas. [delete] is internally recoded as hex \$80 to prevent mixups with \$FF file markers yet still be placeable in the type-ahead key buffer.

To further cloud the issue, end of screen line markers are temporarily placed in machine resident text files as low ASCII, and the rest of the text file pair is coded as high ASCII.

Thus, values of \$00, \$7F, \$80, and \$FF would seem to be disallowed and prevented from ever entering a text file. In reality, it is apparently safe to patch AWIIe so that you can imbed \$80 NULL commands in your text file. You lose an obscure use of [delete] if you try this imbedding. But you should *never* be using [delete] anyway because [delete] is forever, but */open apple/ [—]* can be undone. You also lose the ability to run in a cardless 40 column mode, which nobody ever does anyway. Finally, you must not imbed a NULL into a WPL label.

The patch works by deactivating the two NULL filters already present in the AWIIe code. One filter is involved with the keyboard input routines. The second filter prevents NULLs from being placed in your working text files. The filters are deactivated by branching to continuing code so that NULLs can remain.

After your patch is completed, you can imbed a NULL in your text file by entering a *[V]@[V]* or by using the easier-to-key *[V][2][V]*. Again, we are using WPL notation here, where [V] means *press and hold the CONTROL key, then press and release the V key. Finally, release the CONTROL key.*

Like the classic story of a none-too-bright user who looked and looked but never found a key marked ANY and was thus unable to continue.

The single most important use of NULLs involves superscripting and underlining on older Epson printers. Extra NULLs might also be needed to complete the setup of a special printer card or to activate a modem. The use of NULLs can be made fully automatic by including them in the glossary for the modem or printer in use. Details vary with your choice of hardware.

One gotcha: ProDOS Applewriter 2.0 absolutely and totally forbids any NULLs in its text file, anytime, ever. If you are going to *convert* any AWIIe files that contain NULLs, be sure to change each NULL to a US user separator [—] first. More on this shortly.

## ***The STRETCHIFIER (AWIIe)***

Probably the single most serious legitimate complaint against AWIIe is that special imbedded printer commands are counted as real characters when fill justifying a line. Thus, if you try to do anything intelligent with your printer, you end up with short lines mixed in with fill justified text.

Patch B.2 is called the STRETCHIFIER and greatly eases the shortline problem. Most imbedded commands to most printers begin with an escape command. The STRETCHIFIER scans characters as they are being formatted into a line. Should any escape command be found, the line containing this command is lengthened by two counts. Thus, a two byte escape command fed to an intelligent printer is exactly compensated. Any number of two byte commands in the same line can be fixed similarly.

What if a three byte or higher escape sequence is needed? We have to resort to some skullduggery if our code is to work on any printer or any sequence. What you can do is bank characters ahead of time. Most printers will ignore an `[esc][esc]` sequence, which lengthens the line by four characters but uses two of them for itself. The net result is you have banked two characters, which can be spent by a pair of three byte commands that follow.

To bank a single character, you usually can use an `[esc][esc].@` if you also have done the NULLIFIER patch. The `@` is output as a NULL or ASCII \$00. The NULL burns up a character but normally will be ignored by everything.

Banking of characters sounds awful, but you can include the banking code in with the actual imbedded commands in the glossary for your printer or modem. You can even make the banking fully automatic by using either the glossary or WPL.

Note that nonescape control sequences can also be fixed by pre-banking escape commands. Thus, with enough skullduggery, you should be able to repair exactly almost any imbedded printer command, escape or otherwise. Once such repairs are discovered they can be handled with a single glossary keystroke or done automatically by WPL.

Buried inside the STRETCHIFIER patch is one obscure detail. Word wraparound is always in use during fill justify formatting. What if a word at the end of the line is too long to fit but includes an imbedded escape sequence? To handle this rare possibility, after a line is completed, another short routine eliminates any extra character counts in words that are not to be printed on this particular line. Thus, you first add two counts for each escape sequence on the line. Then you knock off two counts for each escape sequence imbedded in the final word *only* if that word will not fit. This final word and its imbedded commands will be picked up automatically as the first word on the next line.

The STRETCHIFIER borrows space from an essentially useless Volume Verify routine in AWIIe. By defaulting this routine to an immediate subroutine return, room is made for both the STRETCHIFIER and the upcoming CLARIFIER with one byte to spare.



The STRETCHIFIER also includes the upcoming RESTORIFIER patch, so the help menus still work.

Operation of the STRETCHIFIER is essentially invisible and fully automatic on two byte imbedded commands. As we have seen, character banking tricks can be used for three byte escape sequences. Details vary with your choice of intelligent printer, typesetter, or modem. Although you will see the most improvement in the fill justification mode, left justification will also be significantly improved.

## *The CURSIFIER (AWIIe)*

One of the serious limitations of WPL is that it has no way to perform negative or multiresult tests on one cursed character at a time. Although characters can be found and replaced, an IF THEN-ELSE capability would be much more powerful, much faster, and also solves the "where am I?" dilemma that is caused by not being able to tell any one carriage return from another.

One specific example: In an assembler listing to be edited, you might want to tab portions of lines that do *not* begin with an \* or a : symbol. The number of fields to be tabbed varies from line to line, and the tabbing must end on the next carriage return. The tab character can be either an [I] or the first space in a string of spaces. Trying to do all this with stock WPL will drive you up the wall and will be excruciatingly slow. Untabbing before a disk save is even more fun.

There are several roundabout ways to read the cursed character in WPL. Reading this character involves finding the character, bracketing it with a unique set of markers and then doing a .pls string load from memory. The brackets then have to be erased.

This route not only takes forever but takes "forever squared" as your document gets progressively longer. The process gets out of hand totally for anything more than a few hundred characters. Using a disk transfer file instead is equally discouraging on long text files.

Patch B.3 is called the CURSIFIER. It very rapidly puts the cursed character into the WPL \$D string on a [Q]-K command. The old and patently useless Quit routine was diverted for this new feature. Once in the \$D string, you can rapidly test the character in many different ways.

What good is the CURSIFIER? For openers, assembly language editing is done much more easily by AWIIe and WPL than by the rather primitive line editors in most assemblers. Although WPL is great for numbering and renumbering listings, we have seen that tabbing and untabbing of assembly listings cannot be done easily without the CURSIFIER. My *Apple II/IIe Assembly Cookbook* (SAMS #22331) has many more details on this sort of thing.

Few people realize that Applewriter IIe is even better at processing pictures than words. Most plotter commands consist of long text strings, and long text strings are what WPL knows and loves best. Another use of

the CURSIFIER involves hooking up a plotter with a pen that has been replaced by a scanning photocell for automated conversions of input artwork to numeric data bases. A more typical use of the CURSIFIER: It simply scans a document, then makes character by character decisions based on where in the file the module is and what it must accomplish.

## ***An Example***

Look at a sneaky use of the CURSIFIER. One of the most needed AWIIe features is a space-on-disk routine. Program A.9 is called WPL.SPACE ON DISK. It finds the available space remaining on any disk in either drive and does so without hurting the text file sitting in the machine. This module is fairly compact and reasonably fast. Around six seconds are needed to calculate the space remaining on a disk containing a dozen files.

The CURSIFIER is used as a numeric scanner in this routine. WPL enthusiasts will find lots of sneakiness in this module, including a mind-boggling, instant multiply-by-ten code line and creative use of wild cards. Naturally, I will save the details on this one as an exercise for the serious student.

Or maybe I'll hold it for the final exam.

You can actually watch this module in operation by changing the .pnd to a .pyd. Some slop is left in the program for you to play with. It can be sped up and shortened. The formatting possibly can be simplified, and extra code can be added to preserve your exact place in your work file. Have fun.

Although several other space-on-disk routines are kicking around the user groups, note that this one is extremely fast, does not disturb your resident text file, and does not mess with DOS. Note that space on disk is automatically provided on new ProDOS Applewriter 2.0 as a normal part of the [O]-A catalog.

## ***The PATCHIFIER (AWIIe)***

AWIIe's real power is not unleashed until you are able to extend and modify it to suit your own needs. Such things as HIRES graphics dumps, picture processing plotter routines, co-resident assemblers, or integrated file management demand the capability of customizing your code well beyond what you can do with fixed and simple patches.

The PATCHIFIER of Patch B.4 gives you or WPL the ability to modify machine resident AWIIe code at any time for any reason. This patch is both very simple and very elegant. The Verify File command is very rarely used, so it is simply renamed as a Bload Patch. AWIIe handles most of its disk access commands by taking the first word in the DOS menu selection, forcing uppercase and then sending that word to a somewhat modified DOS 3.3e.

When this patch is completed, you or WPL can easily BLOAD anything anywhere in your machine. This is essential for mid document HIRES graphics dumps or for other heavy extensions to the stock AWIIe code.

To make your patch, create a binary file on disk. Your patch can range from a single byte for a POKE by WPL, or a single command sent to an oddball printer card, through a complete overwrite of the entire program that you are patching. Inserting a trailer directly into the filename will relocate the code as it comes off disk.

This patch is deadly. If you gain the ability to modify code, you also gain the ability to destroy code. Improper or unknowing use of patches can completely destroy the integrity of Applewriter IIe, not to mention irrecoverably wiping out valuable text files. Use this powerful tool only at your own risk. Do not use it at all unless you have completely documented the source code you have on hand and are an accomplished machine language programmer.

## ***The LINKIFIER (AWIIe)***

Lots of times you would rather do things your own way.

It sure would be nice to be able to link any custom machine language modules of your own to AWIIe so that they could be called as a subroutine. Such modules would be most handy for HIRES dumps, plotter firmware, speeding up fancy WPL routines, sorts, and much more.

The LINKIFIER of Patch B.5 gives you a way to call your own machine language code as a subroutine when either you or WPL call for a [Q]-H. The original use of [Q]-H was to toggle the data line display. This feature was most needed in older versions of Applewriter, but [esc] does exactly the same thing in AWIIe. Thus, [Q]-H is a leftover that is no longer needed.

All you have to do to link to [Q]-H is put the destination address of the custom code module where the original toggle address used to be. It is also a good idea to change the auxiliary functions menu to spell out exactly what your module is to do.

Note that you can use many different custom modules by combining the PATCHIFIER and the LINKIFIER. Use the PATCHIFIER to put the starting address in the [Q]-H slot, then use the LINKIFIER to call that module when and as needed. As with the PATCHIFIER, the LINKIFIER can be deadly, particularly if you go hooping off to a module that does not exist. The integrity of AWIIe and all of your work files depends entirely on how carefully and precisely you make use of this powerful new tool.

## ***The CLARIFIER***

If you try running an older version of AWIIe on a IIc, you will find the status display line getting trashed and hard to read. In addition, random and usually bizarre changes in the cursor symbol may temporarily occur.



Why does all this happen?

A mouse nest in the IIC character generator causes these compatibility hassles. In a IIE with the old ROMs (before January of 1985), codes of \$40-\$5F appear on the screen as inverse uppercase characters. In a IIC, or a mousified IIE, codes of \$40-\$5F normally appear as mouse text symbols.

Although the new IIE or IIC ROMs offer a way to turn off the mouse text, older Applewriter IIE uses its own internal routines to put characters on the screen and neither knows about nor expects the mouse nest or its controlling firmware.

Applewriter IIE simply guessed wrong in its choice of code for inverse uppercase letters. Any inverse uppercase letter in older AWIIE routines will appear on the screen as a mouse text symbol.

The mouse nest causes severe IIC compatibility problems that show up in far more programs than Apple would care to admit. Apple's sledgehammer repair for this problem is new ProDOS Applewriter 2.0, which is fully compatible with the "old" IIE, the "new" IIE, and the IIC. This solution does not help those of you with older AWIIE code on hand that you wish to use glitch free on a IIC or new IIE or for those of you who prefer to retain DOS 3.3 compatibility.

Patch B.6 is called the CLARIFIER. The CLARIFIER will change any inverse uppercase characters in the status line display so that this character will appear correctly on either an older IIE or a new IIC or new IIE.

The CLARIFIER works by branching to some room cleared away from the Volume Verify code area. It then checks for inverse uppercase in the \$40-\$5F range. These characters are then forced into the alternate inverse uppercase range of \$00-\$1F. The status line should now be clean on either a IIC, an old IIE, or a new IIE. Be sure to make the CLARIFIER patch when you make your IIE chip upgrade.

No attempt was made to fix the occasional and temporary symbol change of the IIC flashing cursor. The patches that I have looked at which are supposed to fix this introduce more problems than the patches solve. Especially sticky is the display of embedded control commands. The temporary change of cursor to a mouse text character happens only when the cursor sits on an uppercase character.

The check mark or whatever that shows up every now and then is kind of cute and not all that distracting. You could eliminate the flashing of the cursor but doing so would introduce serious problems during split screen, embedded character, and search operations.

You could also burn your own "old" character generator to eliminate the problem, but then you no longer could use a mouse on your IIC.

The CURSIFIER must and does include the upcoming RESTORIFIER patch because CURSIFIER overwrites a module that enables the help screens. Although we have shown both E and F version patches, only the F version normally would be run or used in a IIC unless you are up to something *very* strange. A "new" IIE conceivably could use either patch, although failing to include extra memory with your upgrade would be kind of shortsighted.

## ***The RESTORIFIER (AWIIe)***

Usually, it is nice to keep the patched code the same length as the original, so the normal trick is to find modules that you can eliminate inside the code. You then replace these modules with your patches. The Volume Verify and Quit modules are totally useless, so they are the ones that were overwritten with the various patches.

Unfortunately, the Volume Verify routine initializes the slot number for the help screens. Fail to initialize the slot number and your help screen requests give you a nasty I/O ERROR. And no help screens are bad news, especially if beginners or part time users are involved.

Patch B.7 is our RESTORIFIER patch. It is all of one byte long. All this patch does is force feed a slot six ASCII data value into the help screen code. Note that this patch will only work if you are using slot six for your disk drive. Just about everyone uses that slot anyway, so this restriction is no biggie.

That completes our collection of AWIIe patches. Onward and upward to ProDOS 2.0 patches.

## ***ProDOS 2.0 Patches***

As you might expect, the upcoming ProDOS Applewriter 2.0 patches do different things in different ways than the AWIIe patches. An AWIIe patch installed on ProDOS Applewriter 2.0 will work almost as well as it would if you installed it in the middle of Zork® or Visicalc®.

Don't try this. Don't even think about it.

Let's review briefly the advantages of ProDOS Applewriter 2.0 over AWIIe: First, the new version is faster and easier to use. Thankfully, the program is totally unlocked and completely unprotected. You can make as many backup copies as you need. You can also transfer *all* of your files to the hard disk of your choice. You can easily link your own custom machine language modules, any way you care to. Even the source code is capturable. Compatibility with other ProDOS programs, notably Appleworks, is greatly improved.

Secondly, you can now set your right and left screen margins any way you like, moving very close to what-you-see-is-what-you-get processing yet retaining all the power of embedded formatting commands that sophisticated and serious users need. You can now edit spreadsheets up to 240 columns wide.

Thirdly, there's a built in modem that you can use to send and receive text files directly from within the Applewriter environment.

Fourthly, you'll find a "bunch" of minor improvements, including trace over pre-prompted finds and saves, a page/position status option, a space-on-disk display, full IIc or new IIe compatibility, and a few other odds and ends.



Fifthly, the program retains all of the powerful features of the earlier Applewriter programs. The thing that makes Applewriter totally unique is its WPL word processing language, a simple yet devastatingly powerful way to automate practically all word processing operations.

As we have seen, WPL eliminates the need for any separate form letter or mailing list modules. WPL provides a way for you to print all sections of all chapters of an entire book without any intervention. This language lets you pull tricks like printing with true microjustification and proportional spacing (on certain printers only), using a camera-ready print quality improvement technique, producing self prompting glossaries, providing global search and replace across file and disk boundaries, and even handling multiple columns.

Finally, Apple has made upgrading a real bargain. The program lists for \$150 new from your local Apple dealer. However, if you own *any* older copy of Applewriter, you can upgrade for only \$50. To upgrade, you send \$50, the cover from your manual, and the first factory disk of your older Applewriter version to Applewriter Upgrade, Box 306, Half Moon Bay, CA, 94019.

The bottom line is that ProDOS Applewriter 2.0 is a great improvement on an already great word processor. As with any new or revised product, a few loose ends and a few ratty edges can stand some touching up. That is what we will do here. We will show the patch installation process internally to each upcoming patch.

Here are the most requested patches for ProDOS Applewriter 2.0.

## ***PREFIXIFIER (ProDOS 2.0)***

Patch B.8 is the PREFIXIFIER and uniquely solves the prefix hassle. ProDOS *demands* a prefix volume name for any and all ProDOS disks. Volume names are hard to remember and a pain to look up. Instead, Applewriter provides for a Set Prefix option under [O]-H. You can set this prefix to the name of the volume in the desired drive, or you can take a much simpler route and *directly* set this prefix to ,d1 or ,d2.

To eliminate having to set the prefix on a cold boot, just let a STARTUP program do the job for you.

If you do not already have a STARTUP program, you can use the PREFIXIFIER directly. Normally, you use a WPL STARTUP routine to do all sorts of good things, such as loading your favorite self prompting glossary and then loading your stock print constants and tab values. If you already have a STARTUP program, all you have to do is add the single line of Patch B.8 to your existing routine.

Although PREFIXIFIER is not a true code patch, solving the prefix hassle is so important that its solution rightly belongs here.

## ***AIOIFIER (ProDOS 2.0)***

Patch B.9 is our AIOIFIER. When Apple made the upgrade, they very carefully avoided any use of machine language location \$0024 because the program author knew that certain parallel printer cards had a nasty habit of messing with this location. Because the internal workings of ProDOS Applewriter 2.0 use their own screen updating code, such tampering with a screen cursor by an external card is intolerable. Thus, nothing is connected to \$0024 when Applewriter is active. This opens a can of worms for many non-Apple parallel or serial printer cards. Bear in mind that Apple designed this new version to be IIc compatible and thus assumed that slot one and slot two firmware used standard Apple serial design rules. Apple didn't intend for the program to support older third party cards, serial or parallel.

You have to add your own patch if you want to use an older non Apple printer card in a IIe. At first, having to add your own patch sounds mean and nasty.

But think about it for a while.

No matter how many brand-x parallel printer cards the program supported, some company would come up with one more far-out product that would not work.

Applewriter is easily patched, and if you feel strongly about a brand-x card, you have to fix its interface. In general, what you have to do is (1) defeat all video echo, (2) don't assume anything about location \$24 on page zero, (3) make sure the card's cursor is always between the card's internal right and left margins, and (4) don't force carriage returns with the card.

I'll show two specific fixes here, one serial and one parallel.

The AIOIFIER handles the fix for the older AIO serial interface. The AIO symptoms are striping out or ignoring any embedded printer commands while printing the letter part of the embedded command as a real character. The cause is the automatic addition of a space after any control command. That addition happens when both location \$24 and the internal cursor are set to \$00.

The AIOIFIER patch works by jumping to a short custom link. The short custom link turns off the video echo, then sets the AIO right margin to \$FF. Needless to say, each and every brand-x card will need its own custom link, even though all the links will be alike-but-different-somehow. The patch most asked for is discussed . . .

## ***GRAPPLIFIER (ProDOS 2.0)***

Patch B.10 is our GRAPPLIFIER patch. It is intended to cure the Grappler card problems. The usual symptoms are random bursts of 23 spaces inserted every 240 characters or so. The patch is somewhat similar to the AIOIFIER. What the patch does is cancel all Grappler operating modes, including video echo. Then the Grappler internal line length is set

to \$00. Location \$24 on page zero is set to \$01; and the internal cursor is set to \$02. These settings trick the card so that it never outputs extra spaces at random. These commands are repeated immediately before *each* character is sent to the card.

If you have some card other than a Grappler, you can try the AIOIFIER or GRAPPLIFIER to see whether either one will help. Chances are that you will have to disassemble the card's firmware before you can make an intelligent patch.

Note that the GRAPPLIFIER makes the program slightly longer than it used to be. If you assume that your program always ends at \$6020, you should be safe for most any patch needed for most any card.

The dealer-supplied upgrade to ProDOS 2.1 can eliminate certain card problems.

## ***BOOTIFIER (ProDOS 2.0)***

Patch B.11 is the BOOTIFIER. Some people have complained that the help screen and the return prompt which appears on cold boot serves no useful purpose.

If you have only one drive, these bootup features give you the chance to change to your application program before the PRT.SYS and TAB.SYS constants are loaded. This screen and prompt also give beginners a clue to what program they are running and how to access the many help screens.

The BOOTIFIER patch eliminates the first screen and the key prompt. Use this patch only if (1) you have two drives, (2) a novice will *never* access your copy, and (3) you get impatient *very* easily.

## ***NULLIFIER (ProDOS 2.0)***

Patch B.12 is the ProDOS NULLIFIER. This patch is drastically different from the NULLIFIER program and patch for Applewriter IIe and is only needed under very unusual circumstances. A NULL is disallowed in a ProDOS Applewriter 2.0 text file.

Period.

No patches. No fixes. \$00 is reserved as an open-end text file marker.

Certain printers, particularly older Epson models, demand NULLs for such things as stopping underlining or doing superscripts. A compromise is used in ProDOS Applewriter 2.0 that will delight the Epson people and infuriate the Diablo people. In the stock program, a US user separator [—] is substituted any time you need a NULL in your text file. As the program is printed, a NULL is substituted automatically and irrevocably for every US. US is a HMI motion command for daisywheels. People who use daisywheels generally need this command and so do people who use certain modems, particularly Hayes products.



Patch B.12 lets you change the NULL substitute character to something else. If you do not need US user separator commands, leave things the way they are. If you need a US and do not need NULLs, substitute a \$00 for itself. If you must have both a US and a NULL, find some character you do *not* need and use it as a substitute.

## ***GLOSSIFIER (ProDOS 2.0)***

Patch B.13 is the GLOSSIFIER. After a very careful and exhaustive search of the thousands upon thousands of bytes in the Applewriter code, I have found only one that was just plain wrong.

This is it.

If you enter glossary strings by hand in the unmodified code, you can eventually overfill the glossary. No error message is displayed, and the overfill destroys the whole program. This one byte patch carefully checks a glossary entry to make sure it will not overflow the glossary. When a user attempts to make an entry that will overflow the glossary, an alarm is sounded and the overflow is aborted.

## ***CREEPIFIER (ProDOS 2.0)***

Patch B.14 is called the CREEPIFIER. A very minor and innocuous bug tacks an extra space at the end of top line and bottom line entries as they are printed.

So what? Big deal.

You would not believe how much grief this extra space has caused how many people making helpline calls. *If* you set Applewriter's right margin to 80 and *if* you set your printer card right margin to 80 and *if* you suppress the top and bottom lines on the first page, you get page creep. With page creep, the paper perforations seem to walk up or down the page on successive pages.

A related symptom is that the .tm top margin and the .bm bottom margin always seem one line more than they should be. An obvious cure is to use .rm78 in Applewriter, but nobody ever does. Apparently, using .rm78 never occurs to anyone. So Patch B.14 corrects what should be a totally negligible bug. The patch works by entering a subroutine in its middle so that the sub prints only a single carriage return rather than a space followed by a carriage return.

## ***SCRUNCHIFIER (ProDOS 2.0)***

Patch B.15 is the SCRUNCHIFIER. Applewriter has an early heritage of 40 column displays. This results in the prompting menus taking up most of

the vertical screen. This is attractive and no big deal on the [P] and [Q] prompts. However, on the [O] ProDOS options prompt, not only does the menu take more than half the screen, but the menu also scrolls a previous catalog or whatever else you just did off the screen.

The SCRUNCHIFIER will give you a two line ProDOS option menu that leaves much more of a previous catalog or a previous task on the screen. As a side benefit, the SCRUNCHIFIER frees some 100 bytes of code to make room for the upcoming STRETCHIFIER and CURSIFIER patches.

The patch does make this menu look different than the others and purposely misspells *Subdirectory* so that it will fit compactly where it belongs.

## ***STRETCHIFIER (ProDOS 2.0)***

Patch B.16 is the STRETCHIFIER. This patch is alike-but-different-somehow from the similar STRETCHIFIER patch for AWIIe. The stock Applewriter printer routines will count any embedded printer commands as real printing characters. Thus, if you use the underliner once on your printer, your line will end up four characters short. If you use the underliner several times, the line ends up ludicrously short.

This STRETCHIFIER, as the earlier version, works by adding two counts for every escape character that the patch finds and actually uses when formatting a printed line. This will compensate exactly for any embedded printer command that consists of an escape followed by a single letter.

Banking can be used to handle embedded multiple letter escape commands. For instance, you can use an *[esc][esc]* to bank two characters or an *[esc]* [null substitute] to bank one character. Both of these strings usually will be ignored by most printers. You can easily add these banking commands to your self prompting printer glossary so that they will be handled invisibly and automatically.

## ***PROMPTIFIER (ProDOS 2.0)***

Patch B.17 is called a PROMPTIFIER. As we've seen, it's very simple to have a glossary self prompt so that you get an instant on screen help menu when you need it. Self prompting sure beats memorizing commands or taping notes to your Apple.

Older self prompting glossaries assumed that the Load to Screen command was a backslash. The new version of the program loads to screen only on a variable UT valve. Thus, if you have no UT underline token, you will end up putting into the middle of your text file what should have gone only onto the screen. Older WPL programs will be similarly affected. The PROMPTIFIER puts things back the way they were. Once patched, the backslash becomes the unconditional Load to Screen command.

## ***CURSIFIER (ProDOS 2.0)***

Patch B.18 is called the CURSIFIER. This one shows you how you can easily add a single custom code module of your own. CURSIFIER will put a copy of whatever the cursor is pointing at in the \$D string, which lets you scan a file for certain characters or groupings. It also can greatly increase the speed of certain WPL routines. The CURSIFIER also forces the cursed character into low ASCII, eliminating a subtle start-of-line marker hassle.

This module also substitutes \$0A line feeds for \$0D carriage returns. This last stunt lets you search for or find carriage returns, a process that is extremely tricky otherwise. What happens is this: Option [Q]-H to toggle the status line is left over from three generations back. [esc] does the same thing, so you are free to grab [Q]-H and do anything you want to with it.

Your patch has three steps. First, you have to put the correct starting address of your custom routine into the program where the link for toggling the display was. Secondly, you have to relabel the screen message for [Q]-H. Finally, you have to install your patch somewhere.

## ***Wrap-Up***

And that just about wraps up this chapter. You will find the AWIIe patches ready to go on the AWIIe companion diskette. You will also find the ProDOS 2.0 patches ready to go on, of all places, the ProDOS 2.0 companion diskette.

I purposely have shown only ProDOS Applewriter 2.0 patches for the AWD.SYS program, which is intended for a IIc in 80 columns or else an old or new IIe with extended memory. You can work up similar patches for AWC.SYS for a short 64K IIe or AWB.SYS (for a IIc in its 40 column mode).

Contact the helpline directly for patches for the Version 2.1 upgrade. In general these patches lie a few bytes beyond the corresponding 2.0 patch.

Have fun exploring these patches and be sure to let us know about others you want to see. Yes, we are working on HIRES dumps, laser printer interfaces, and Appleworks transfers.

You users can go away now. Us hackers are now going to get into the fun stuff as soon as we start the next chapter . . .

---

# 7

---

## **Tearing into ProDOS Applewriter Version 2.0**

This complete, thorough, and detailed  
disassembly script  
will show you exactly  
how ProDOS Applewriter 2.0 works  
and how and where it can be modified . . .

---





A new ProDOS-based version of Applewriter was introduced by Apple Computer in November of 1984. Its many advantages include full IIe and IIc compatibility, easier hard disk interfacing, better compatibility with Appleworks, and significantly faster disk access.

Important new features include arbitrary setting of your screen margins instead of using a fixed 80 character screen. This feature means you have much more of what-you-see-is-what-you-get and can easily edit spreadsheets or data bases up to 240 columns wide.

You can use the limited telecommunications capability to send or receive text files via a modem from directly *within* Applewriter 2.0. An optional page/position display tells you exactly where you are in a multipage document. You can optionally suppress headers and footers on your first page. Remaining space on disk is included in your catalog options.

Best of all, the program is completely unlocked, unprotected, copyable, and fully open.

The program has lots of bugs. Some are new and some are left over from before. Although patching is simple, the 2.0 patches are very much card dependent. One major bug involves initializing a disk. You cannot do so without trashing both your glossary and WPL. Support and use of parallel printer cards and parallel printers can be extremely tricky. We have already seen fixes in the preceding chapter's AIOIFIER and GRAPPLIFIER patches. The shortline problem remains, although the STRETCHIFIER patch described in Chapter 6 gives a quick and easy fix.

An attempt to fix the NULL problem opened a real can of worms. A user separator [—] is irrevocably used as a substitute NULL character, which gives you a way to subscript and superscript on an old Epson. Unfortunately, this substitution deprives you of any HMI commands on most daisywheels and royally fouls up certain modem cards. The NULLIFIER patch described in Chapter 6 cures this hassle.

If you use ProDOS Applewriter version 2.0 on a IIe with an intelligent third-party card, you can almost be certain to expect strange and wondrous things happening to your printout, such as random bursts of spaces distributed helter-skelter in the printout or total suppression of imbedded printer control commands.

A ProDOS Applewriter 2.1 upgrade is available for Apple. This upgrade cures some third party card problems.

ProDOS Applewriter also has several minor bugs, one of which gives you the ability to destructively overload a glossary. This last bug is cured by the GLOSSIFIER patch in Chapter 6.

All support of Applewriter 1.0 and 1.1 has been dropped. Text files written under more recent DOS 3.3 or 3.3e versions of Applewriter are fully compatible except that you must use the CONVERT feature of ProDOS to transfer the files between operating systems. A switch from DOS 3.3e's high ASCII to ProDOS's low ASCII is also taken care of by CONVERT. Glossaries written under Applewriter IIe are compatible, although those written under earlier code may not be owing to a different treatment of [V] in pre-IIe code.

Several warnings on CONVERT. This finder routine sometimes gets sick on longer text files, particularly those that exceed 31,000 characters. If you have problems, try splitting the file in half and then convert both halves. Watch your filenames carefully. Only numbers, periods, and uppercase letters are allowed in a ProDOS filename. The maximum length of a filename is only 16 characters.

Be extremely wary of long filenames. If you convert MY REALLY NEAT STUFF 1.0 and then convert MY REALLY NEAT STUFF 2.0, *both* files will be converted to an *identical* ProDOS filename of MY.REALLY.NEAT.S. Needless to say, several files with identical names cause all sorts of ungood nasties to occur.

Print constants and tab files are *not* compatible from earlier disks because the tab now goes out to 240 characters and new modem goodies are now saved as print constants. You have to reenter these files by hand.

Note that the name sequence of these files has also been reversed. What used to be PRT.MYSTUFF is now MYSTUFF.PRT. To help you avoid mixups, the .PRT part still is automatically tacked on by the program free of charge.

The bottom line: major improvements to an already exceptional word processor. The new program lists for \$150 from your local Apple dealer. A bargain priced upgrade is also available. To upgrade, send your first factory disk from your older Applewriter, the cover off your manual, and \$50 to . . .

APPLEWRITER  
UPGRADE  
Box 306  
Half Moon Bay CA, 94019

As to what bad thing happened to your manual cover in that (flood) (fire) (tornado) (termite) (goat) (butterscotch) (KGB) (laundry) (whatever) incident and the sob story you are planning to tell, forget it. The Apple people have heard all the excuses.

No tickee, no washee.

Be sure to back up your old copy before you send it in. On Applewriter IIe, Copy II+ works fine after a parameter change of 10:96.

I have already done a complete disassembly of the older Applewriter IIe, which you will find in my *Enhancing Your Apple IIe*, Volume II (SAMS #22425) and in my disk-based AWIIe Toolkit (sides 1-8) from Synergetics. What I'll do in the rest of this chapter is give the same disassembly treatment to ProDOS Applewriter 2.0.

Applewriter actually comes in three new versions, called AWB.SYS, AWC.SYS, and AWD.SYS. These files are all present on your Applewriter 2.0 master disk. The B version is for a IIc switched to 40 columns; the C version is for the 64K IIe; and the D version is for the 128K IIe, for the IIc switched to 80 columns, and for all future Apple products having 128K and a 40/80 switch in the 80 position. Should you want 40 columns on a IIe, you are asked to fake it by manually setting your screen margins. Thus the D version is the heavy, used by most of the people most of the time.

We will limit our analysis here to the D version of ProDOS Applewriter 2.0. If you are going to interface anything else to your word processor, the D version is the best choice at this writing because it has a reasonable amount of uncommitted RAM left and supports the open slots of the IIe.

If you are using a version other than D, apply the following ideas to analyzing any word processor or machine language program on any system you choose. You will find both the older Applewriter 2.0 and Applewriter IIe programs to be alike-but-different-somehow.

When compared to Applewriter IIe, the new code is longer. This is partially because of the new features but mostly because of the memory-hogging ProDOS links. As newer versions of Applewriter become available, we will keep you informed of the latest information as best we can.

## ***Analyzing ProDOS Applewriter 2.0***

Two warnings before we begin. First, everything here is unofficial and unsanctioned. What you see here is just a use of my tearing method on my copy of the program's AWD.SYS version.

In some cases, I've had to make a guess or two in understanding fuzzy parts of the code. In other places I may have missed something obvious. In yet other places I could easily be just plain wrong. Second, the tearing applies only to the D version of ProDOS Applewriter 2.0, circa Fall, 1984. Any changes at all to the program or any use of a different version will alter many of the program locations.



One thing that becomes obvious when you tear into this code is the delicate balance involved in designing a major word processor. People want a word processor that is cheap, very fast, powerful, easy to learn, and has a large work space and many different features. People also want a program that produces an exact screen image and still supports all known features of all known printers and all typesetting machines in all known fonts, present or future, in as many languages as possible.

All of these demands fight each other six ways from Sunday.

Much of the following analysis is based on my tearing method, which first appeared in *Enhancing Your Apple II/IIe*, Volume I, (SAMS #21822). The tearing method is an astonishingly fast and easy way to disassemble and analyze someone else's machine language code.

Let's review how . . .

### To Analyze a Heavy Program

---

0. Use the program till you know it cold and have *completely* mastered its operation.
1. Tear the program apart.
2. Study the memory map.
3. Find out how each file works.
4. Find the ProDOS MLI hooks.
5. Learn all uses of page zero.
6. Master the low level subs.
7. Attack high level entry points.
8. Try simple mods.
9. Capture the source code.

Step zero is far and away the most important. You cannot possibly understand *any* program if you do not use it daily and continuously until you *completely* understand what the program does and how it does it. Omit step zero and nothing that follows will make any sense. The best way to understand the other steps is to do them in order, by yourself and by hand.

As a reminder, we are assuming you have either an Apple IIe with 128K split as a main 64K RAM bank and an auxiliary 64K RAM bank plugged in slot zero or a IIc switched to 80 columns. We also assume you are under stock ProDOS.

We will only briefly touch on the booting process here. On a boot, a totally standard version of ProDOS is installed in high main RAM. Text files created by Applewriter under ProDOS are completely compatible with any other ProDOS text file. Text files also may be transferred to and from DOS 3.3e by using the CONVERT feature of ProDOS. This is done much as MUFFIN and DEMUFFIN were once used to get between

running old 15 sector DOS 3.2 and 16 sector DOS 3.3.

Long ago and far away.

The DOS boot process loads and runs a machine language program called `AW.SYSTEM`, which will be booted automatically if this program is listed as the first `.SYS` file on the disk. `AW.SYSTEM`, in turn, analyzes the Apple to see what it is and what is connected where.

If you do not have an Apple IIc or IIe, an error message appears that tells you the bad news, hangs the machine, and then kicks sand in your face.

No, there is no sane way to run any newer version of Applewriter on a II or II+ or any valid reason even to want to. If you have *no* 64K extended memory card in slot zero of a IIe, a machine language program called `AWC.SYS` is loaded and run. This C version gives you only 22K of main text, besides cramming the IIe to the gills.

If you have a IIc switched to 40 columns, `AWB.SYS` is installed and run. If you have a IIc switched to 80 columns or a 64K extended memory card in slot zero of a IIe, `AWD.SYS` is booted and run, starting at location hex \$2000. The D version, which we will study here, gives you 48K worth of main text file and has lots of space left for your own customizing.

The boot code operation and disassembly script is summarized in Listing C.1.

All of these modules are easily loaded, analyzed, modified, or saved through ordinary ProDOS. Everything is up front. No sneakiness or black magic is involved.

As an example, to view `AWD.SYS`, boot your ProDOS master disk, then do a `BLOAD AWD.SYS, A$2000, E$6020, TSYS, D2`. To save an altered version (to a well labeled *new* disk!) do a `BSAVE AWD.SYS, A$2000, E$6020, TSYS, D2`.

Let's briefly review the command features new to ProDOS: the `TSYS` command `BLOADs` or `BSAVEs` any `.SYSTEM`, or `$FF` i.d. file. You can similarly use `TTXT` or `TBAS` for other sneaky uses. This `Type` command binarily loads or saves in binary any part of any type of file anywhere in memory. The `E` command specifies the *end* of a binary image rather than its `L` length.

In the above example, we assume you have *not* significantly lengthened the code before saving it. Longer code will, of course, have a larger `L` or `E` trailer. The original end of `AWD.SYS` is at `$5FFF`. I have stretched the end to `$6020` to make room for a printer patch or other short code block. See the previous chapter for more details.

Another way to capture the program is to use an absolute reset modified IIc or IIe. Packages to return total reset control of a IIc or a new or old IIe back to you are available directly from Synergetics.

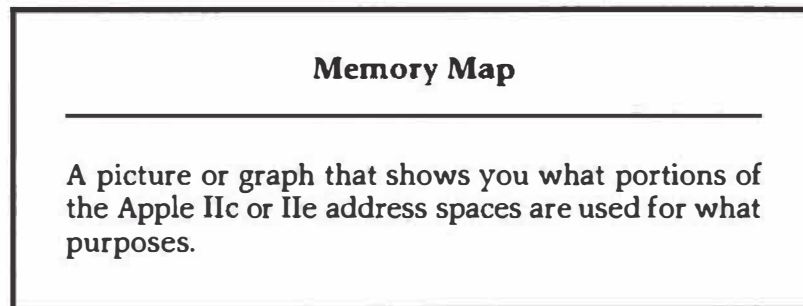
Note that the `AW.SYSTEM` code is easily customized.

This lets you do things like turn modems on, download custom character sets or proportional tables to your printer, initialize custom I/O cards, and so on. But note also that the main code in ProDOS Applewriter 2.0 very carefully disconnects and sets aside anything that was plugged in

slot three. Only on a [Q]-J quit does anything plugged into slot 3 get reconnected. Thus, linking a third party, slot three video or extended text card to ProDOS Applewriter 2.0 would be extremely difficult unless something other than a preboot is used.

## The Memory Maps

The biggest step towards understanding heavy code is to find what sits where in the machine. One or more *memory maps* does the trick.



If you do not know where everything sits in the machine or do not thoroughly understand how the various parts are intended to work together, you have no hope of going any further.

Getting a memory map with exactly the right amount of detail is often a real hassle. I like to use *simplified* memory maps that show only the big picture and separate *detailed* memory maps or lists that give you all the gory details down to every use of every last bit. Figure 7.1 shows the simplified memory map for the D version of ProDOS Applewriter 2.0.

As a reminder, I am assuming that you have a 128K machine split into a main RAM bank of 64K and an auxiliary RAM bank of 64K, an 80 column display, and a fully stock ProDOS operating environment.

No use is made of any monitor ROM in the machine by the actual word processing program. All key entry, screen display, sound effects, I/O, time delays, etc. are handled internally with custom routines. This allows such features as horizontal and vertical scrolling or 64 character type-ahead buffering for both the keyboard and modem.

ProDOS is loaded into high main RAM during booting. This is the standard ProDOS location from \$D000 FFFF. Only high main RAM is normally used. High ROM (the monitor and Applesoft routines) and auxiliary high RAM (usually empty) are neither used nor accessed. The actual word processing program code is also loaded into main RAM. The working code sits between \$2000 and \$554C. Some reference files that consist mainly of screen messages and address pointer tables are towed along and follow the working code between \$554D and \$5FFF.



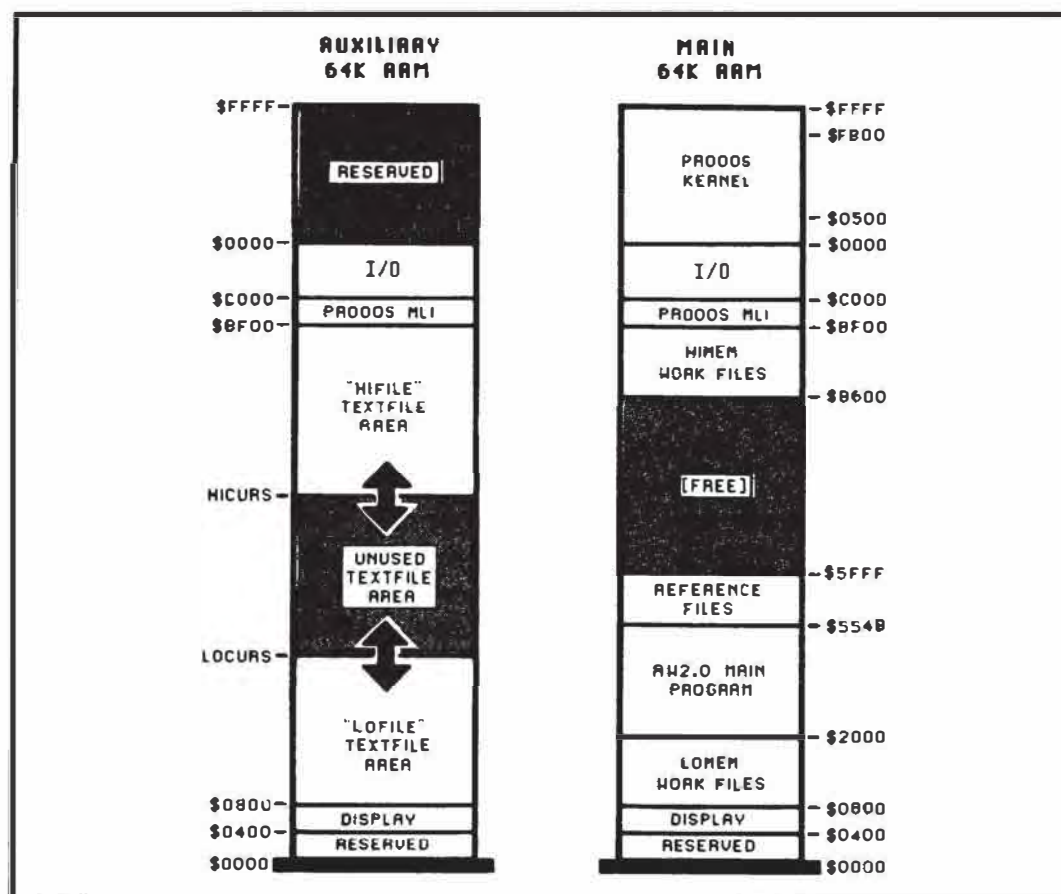


Fig. 7.1. Simplified memory map of an Apple IIc or 128K Apple IIe that is running the AWD.SYS version of ProDOS Appewriter 2.0.

Many work files are needed by this program. The necessary work files include the glossary buffer, two deletion buffers, the WPL program storage area, and many others, as we will see.

The work files are stashed in two areas in main RAM. The *low* work files are stashed from \$0800-\$1FFF. The *high* work files are stashed from \$B700-\$BEFF. The ProDOS interface, or MLI, sits between \$BF00 and \$BFFF. The main RAM area from \$0000 through \$0400 is also used as a work file area. This area includes the important pointers, counters, stashes, and flags on page zero; some memory management and a stack on page one; and some additional work files and links on pages two and three.

Most of the work files are *not* loaded in the machine. They are initialized and then used as the program is run. As usual, the *even* 80 column characters are stashed in main RAM from \$0400 through \$07FF, and the *odd* 80 column characters are stashed in auxiliary RAM in the same address range. Also as usual, \$C000 through \$CFFF is reserved for I/O space uses.

A big chunk of unused RAM sits between \$6000 and \$B6FF in main RAM. This chunk is very lonely and is crying for your attention and use. What can you cram into 21,984 bytes these days?

Turning to the 64K auxiliary RAM, the auxiliary page zero and stack area are not used. Because high auxiliary RAM is switched simultaneously with page zero, all 16K of the high auxiliary RAM is also unused.

Messing with the alternate page zero and alternate high RAM gets tricky fast and should not be attempted by any but the most gonzo hacker.

The text file area that holds the words you want to process takes up the bulk of auxiliary RAM. Your text file area goes from \$0800 through \$BEFF and gives you room for some 46,845 characters at once. As with main RAM, auxiliary RAM locations \$BF00 through BFFF are reserved for a ProDOS interface MLI.

Although Applewriter 2.0 does not directly use this alternate memory MLI, it is preserved and allows Applewriter to transfer control to and from another ProDOS system program in an integrated environment.

Generally what happens is this: The main program puts characters into or removes them from the text file area and provides all the usual word processing functions. The program gets these characters from you at the keyboard; from DOS as text files; from the text file itself for clones and copies; or from special files such as the glossary, the WPL program, your telecommunicating modem, or the deletion buffers. When finished, the main program saves the contents of the text file area to disk or dumps those contents to a printer or modem.

We will be giving you two levels of additional detail on most of these program areas. In the text that follows, we will show you generally what each area is up to. In the various listings, you will find the extreme detail needed for a complete analysis.

Time now to look at . . .

## ***How Each File Works***

Dozens of different files are used in this program. Once you find where the files sit and what they do, you are well on your way to understanding just how ProDOS Applewriter 2.0 handles its various tasks.

We can break those files into five areas: the *text file* area, the *low work file* area, the *high work file* area, the *internal file* area, and finally, the *reference file* area.

Text files hold the words to be processed. As we will see, the program uses a pair of text files in order to dramatically speed up character insertion and deletion.

Work files hold things outside of the program code that are fairly likely to be changed. These things includes all the page zero stuff, the glossary, the WPL file, print programming commands, the tab image file, the top and bottom line buffers, and much more.



Internal files are stuffed inside the working code. Most often, these files are involved with ProDOS interfacing MLI modules or for uses as local "working registers" in absolute memory. You have to pay very close attention to these internal files because you will want to bypass them when you capture your own source code. Otherwise, you will get aliasing and starting off on the wrong foot problems, not to mention illegal op codes.

Reference files hold things that are unlikely to change, which includes screen prompts, address pointers, error messages, DOS commands, and so on.

### ProDOS Applewriter 2.0 File Areas

1. Text File Area  
Holds the words to be processed
2. Work File Areas  
Hold things often changed
3. Internal File Area  
Holds local stashes and ProDOS links
4. Reference File Area  
Holds things seldom changed

Let's check into these file areas one at a time . . .

## Text File Area

The text file area is the single largest and most important. It holds the words to be processed. This area lies in auxiliary RAM from \$0800 through \$BEFF, a total of 46,847 locations. Allowing for the two \$FF end markers, you have a remainder of 46,844 characters. This is the number you see as the MEM prompt on program bootup.

Figure 7.2 shows us how this text file area is managed.

One very sticky problem in word processing involves making your code run so fast that it never gets very far behind your typing. For example, suppose that your cursor is sitting in the middle of a single long text file. On each key entry, you would have to move everything from where your cursor is sitting up or down a character in memory, which could involve tens of thousands of characters.

All of which is bound to be ridiculously slow.

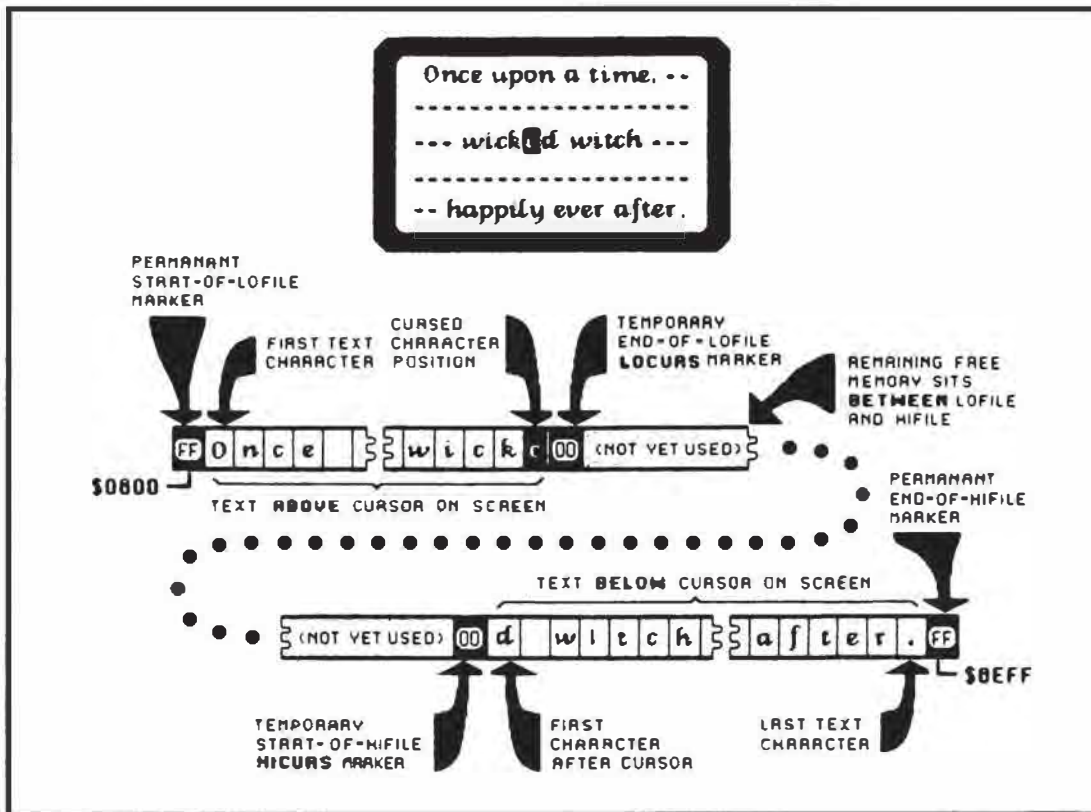


Fig. 7.2. To avoid moving anything on an insertion or deletion, the Applewriter 2.0 screen often displays two files: The RAM LOFILE holds characters above the cursor, and the RAM HIFILE holds characters below the cursor.

So here is . . .

### The Secret to a Fast Word Processing Program

During any fast typing mode, *never* move any character that is already sitting in memory.

Sounds obvious enough, but ignoring this rule is what makes so many competing programs so pitifully slow.

Now for the secret to ProDOS Applewriter 2.0 and a brilliant way to avoid moving things around all the time: Use two separate text file areas, keeping the available free space remaining *between* the two files!

Returning to Figure 7.2, we see a file that I call LOFILE, which holds everything *above* the cursor on the screen back to the beginning of the text.

A file that I call HIFILE holds everything *below* the cursor on the screen forward to the end of the text.

During normal character entry or insertion, you simply add things to the end of LOFILE. Because HIFILE is usually far above LOFILE in RAM, no difference exists between insertion and entry.

To delete, you simply knock characters off the top of LOFILE, again without disturbing HIFILE. Nothing but a single character need be entered or removed from the file for most fast typing needs.

The files do get back together every now and then. For instance, on an [E] command to go to the end of the text, everything is moved back into LOFILE. Your *entire* file now starts at the beginning of the file, and you are free to add to the open end of LOFILE with new characters.

To move everything in a long text file from HIFILE to LOFILE takes the better part of a second, but you make such moves only rarely. You probably would not want to continuously type [E] commands at a 100 word-per-minute rate. In fact, you are usually ready for a brief psychic break when you do an [E].

Similarly, if you do a [B] to get to the beginning of a file, everything is moved to HIFILE. If you add characters from the file's beginning, they go into LOFILE. Once again, nothing in memory needs to be moved during fast entry modes, even on an insertion or deletion.

Meanwhile, fancy things are going on back on the screen. The screen shows copies of pieces of HIFILE and LOFILE. The program magically splices them together as needed to con you into thinking you are looking at one continuous file, showing only the characters between the margins you have selected. Only whole words are shown in the wraparound mode.

Should you be using a right margin wider than 80, a special offset value is added to determine the screen's left margin as compared to the text left margin. If the difference between your right and left margins exceeds 78 characters, special trip points are set separately to provide horizontal scrolling. These trip points are 12 characters in from the left and right sides of the screen.

The screen is usually updated on each character entry, but note that far fewer characters are on the screen than are usually stashed in the text file area. Thus, this screen updating can be done fairly quickly. Updating an insertion does take somewhat longer than updating an addition.

Applewriter puts special marks near the start of each screen line to speed up the update process. These marks are calculated only when they change instead of during each and every screen update.

The *bottom* of LOFILE at \$0800 and the *top* of HIFILE at \$BEFF are always identified by \$FF markers. They tell various service routines, such as the searches and finds, when these routines get to the beginning or the end of either LOFILE or HIFILE. The open end of each file is marked with a \$00 marker, which is the *top* of LOFILE and the *bottom* of HIFILE. These \$00 markers are why NULLs are forbidden in your text files. See the NULLIFIER patch back in Chapter 6 for more details.



The open character on the screen sits at the LOFILE \$00 marker. Should a character be entered, it replaces the \$00. Then a new \$00 marker is added one byte beyond the replacement character for the \$00 marker. The LOFILE pointer is also incremented.

Note that the flashing cursor points one character beyond the open \$00 marker. Thus, the cursor actually points to the first character in HIFILE. The \$FF limit markers and \$00 present end markers are reserved characters. You are not allowed (and are prevented from) entering these characters into your text file. Characters of \$7F and \$80 also are not allowed.

Note that extra zeros are permitted in the unused memory space between LOFILE and HIFILE. Only the *first* \$00 on the way up through LOFILE or the *first* \$00 on the way down through HIFILE matter. If you want to eliminate a string of text you simply put a \$00 at one end, which saves having to ever erase bunches of memory.

A cursor is produced by taking the first character in HIFILE and alternating its screen display between normal and inverse, following a software loop that causes apparent flashing. By the way, the alternate dual-case character set used does not have a hardware flasher available.

Most of the characters in the file are standard low ASCII. To simplify screen updates and word wraparound, the character before the start of each screen line is set to a high ASCII character. This unique approach does screen formatting calculations only once instead of applying special treatment on every screen update. The high ASCII marker usually is the *last* character or carriage return on any screen line. Note that the last character on any one line is always one character preceding the start of the next line. Remember that low ASCII characters have their most significant bit, or MSB, cleared to zero. High ASCII characters have their MSB set to one.

Although high ASCII is traditionally used for Apple screen characters and for older DOS 3.3e text files, low ASCII is used inside ProDOS based text files. Low ASCII is more standard outside the Apple world as well.

As a reminder, characters of \$00, \$7F, \$80, and \$FF are normally reserved and must not be placed in a text file.

In certain ways, then, ProDOS Applewriter 2.0 behaves as a *line oriented* word processor because the program remembers exactly what each display screen line should look like at all times. When text files are saved to disk, all the saved characters are forced to low ASCII. Thus, ProDOS Applewriter 2.0 files are easily exchanged with any program on any computer that can recognize a standard text file full of standard low ASCII characters. This is crucial for typesetting, fancy print formatting, transfers to other computers, and for modem communications.

To review, the 47K text file area usually holds two files called LOFILE and HIFILE. LOFILE holds everything *above* the screen cursor and HIFILE holds everything *below* the screen cursor. These dual files prevent having

to move things around during insertions and deletions and are the keys to acceptable word processing speed.

The beginning of your text at the bottom of LOFILE is marked with an \$FF marker. The end of your text at the top of HIFILE is also marked with an \$FF marker. The open ends of LOFILE and HIFILE are identified with \$00 markers. These two open ends face each other with all of the remaining memory space between them.

The cursed character is the first one at the bottom of HIFILE. Text is added to the top of LOFILE during entry and insertion. During deletion, text at the top of LOFILE is replaced with \$00 markers.

Every now and then, LOFILE and HIFILE are merged together, such as with a [B] that puts everything in HIFILE or an [E] which puts everything back down in LOFILE.

As one fills the other empties.

More often than not, during normal text entry, everything is in LOFILE and you are adding to the *open* end of LOFILE. Text is entered as low ASCII. All characters are allowed except for ASCII codes \$00, \$7F, \$80, and \$FF. A possible conflict with [delete] is handled by temporarily recoding delete as \$80 or as a high ASCII NULL command. This \$80 never reaches the text file because this recoding is filtered by the file entry routines.

The end of each screen line is held as a high ASCII character. This marker provides automatic word wraparound and needs a single calculation rather than special processing on each and every screen update.

Incidentally, defeating word wraparound (no whole word breaks) is only allowed on a fixed screen of 78 or fewer columns. You lose this feature when you custom set your own wider left and right screen margins. Put another way, continuous character display is permitted only when RM-LM is 78 or less. Otherwise, whole word breaks are required.

Be sure you thoroughly understand how LOFILE and HIFILE operate, for they are the keys to understanding the entire program and everything that follows.

## ***The Low Work File Area***

ProDOS Applewriter 2.0 splits its work files into two areas. The first or *low* work file area runs from \$0800 to \$1FFF. The second or *high* work file area runs from \$B600 to \$BEFF. Stuff that is common to the older programs tends to stay in the low work files. ProDOS buffers, the wider tab displays, and similar items have been added to the high work file area.

A memory map of the low work file area is shown in Figure 7.3A.

A detailed low work file breakdown appears as Listing C.2.

Let's look briefly at what these files are and what they do. Starting at the bottom, much of page zero is reserved to hold pointers, counters, stashes, and flags. Page zero is so important that we will reserve the next section for it.

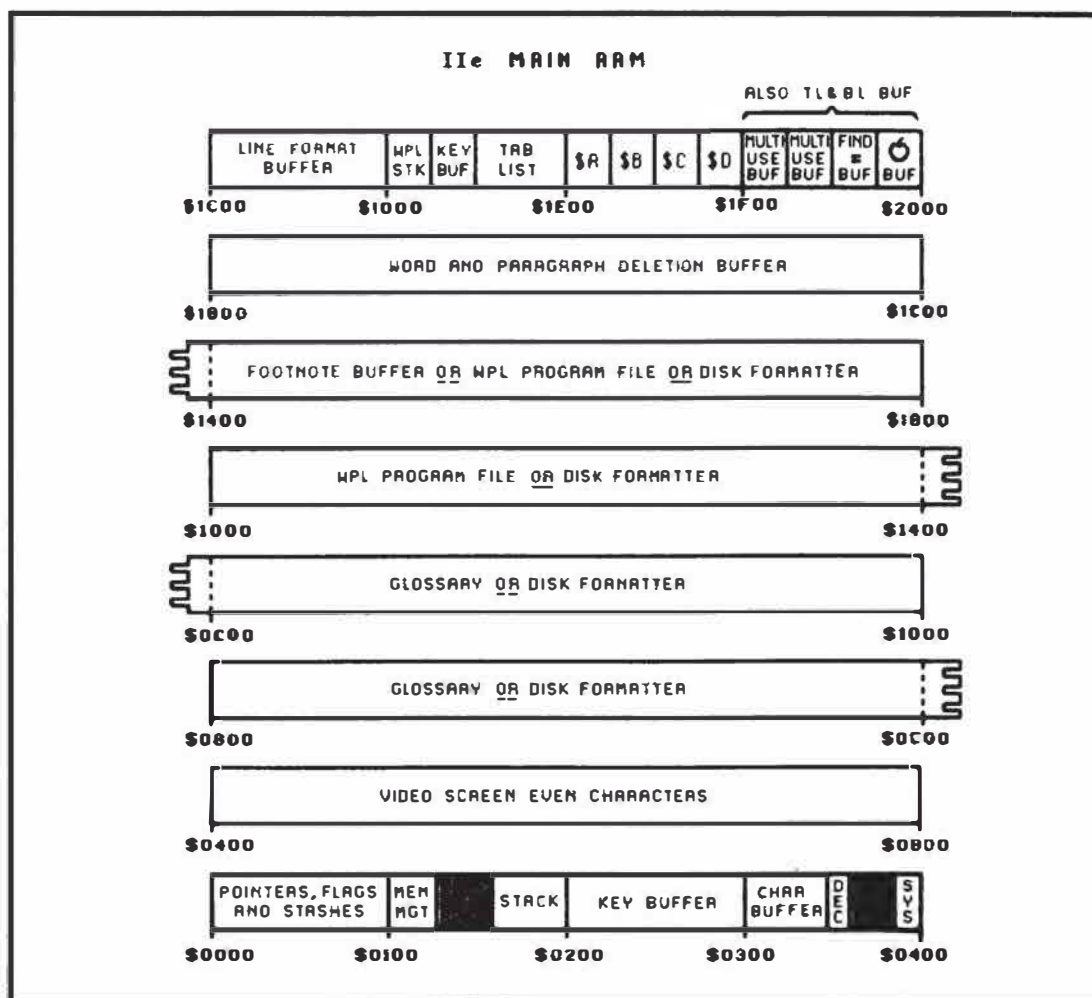


Fig. 7.3A. Simplified memory map of Applewriter 2.0 low memory work files.

The area from \$0100 through \$013E holds the memory management code. Note that main RAM page zero and page one are always used in this program. These memory pages do *not* change on a switch between main and auxiliary RAM. Only RAM memory locations from \$0200 BFFF switch on a change between main and auxiliary RAM.

The memory management code lets you read the text file from a screen update pointer, a low cursor pointer that I call LOCURS, a high cursor pointer that I call HICURS, a printer pointer, an auxiliary pointer, and finally, access a special routine that backs a screen pointer to the beginning of a screen line. By placing the memory management code on page one, either main or auxiliary memory can be accessed as needed.

Remember that your program and the work files sit in main memory and the text files sit in auxiliary memory. The high end of page one holds the stack. As usual, the stack starts at \$01FF and builds down. The stack is short enough that it never crashes down into the memory management code. The stack holds subroutine return addresses. It also sees uses as a temporary value stash. The stack is also temporarily used to hold



addresses for indirect jumps using the forced subroutine return method of option picking.

The keybuffer extends from \$0200 through \$02FF and holds string values and key commands that are entered from the keyboard. The keybuffer is sometimes also used as a temporary work area, as a formatting buffer, or to hold an old character string for later reuse.

Continuing upward through the work files in main RAM, the bottom half of page three is used for the single character swallow buffer. This buffer is controlled by the right or left arrow key in combination with [open apple]. Note that the swallow buffer is separate from the combined word and paragraph deletion buffer higher in the memory.

The usual hooks appear on the top of page three, starting at \$03D0. Most hooks vector to a warm restart of the program at \$20B4.

Let's breeze on by pages four through seven because they are the *even* characters of the text screen. The companion *odd* characters of the text screen are stashed on pages four through seven of the auxiliary memory.

The area from \$0800 up through \$0FFF is normally used as a glossary buffer. Glossary strings build up in memory and end with a carriage return. The last carriage return of the last entry is followed by one or more \$00 end markers. This same memory area is also borrowed by the ProDOS formatting code any time you init a new disk. Note that the glossary is destructively overwritten by this init process.

Moving right along, the work file from \$1000 through \$17FF is a 2K area that can have three possible uses. This entire area normally holds a 2K WPL program file. Should footnotes be needed, a 1K footnote file from \$1400-17FF is usurped from the top of the WPL program area. If you use footnotes, you are only allowed to have a maximum WPL program length of 1K, sitting from \$1000 13FF. As with the glossary, both the WPL program area and the footnote area are destructively overwritten any time you format a disk.

Next are 1024 locations ranging from \$1800 through \$1BFF. These are set aside for the word and paragraph deletion buffers, activated by [W] or [X].

On a deletion, the stuff to be saved is tacked onto the end of anything previously saved. On restoration, the restored stuff is read until a space or a carriage return is found.

The pointers to this deletion buffer go round and round. A separate counter on page zero keeps track of overflow. All of the single page from \$1600 through \$16FF is set up as a line justification buffer. This page is used separately to format the top and bottom lines as well as being useful during search and replace activities.

All of the preceding brings us to \$1D00. Here you'll find the WPL stack that resides from \$1D00-1D3F. This stack holds return addresses for any WPL subroutines in use. A total of 32 subroutine calls are allowed. The type-ahead buffer follows. This area saves up to 64 keystrokes should the typist get ahead of processing. Two pointers access this file on a round and round basis.

One fills and the other empties.

Any [open apple] or [closed apple] commands are separately saved in a second 64 character buffer higher in the work file area. Note that you must save both the pressed key and the state of the [open apple] and [closed apple] keys for a later recovery. Otherwise, certain key combinations will end up doing the wrong things.

Note also that the modem has its own type-ahead buffer which is internal to the main code. This buffer is useful when characters are being received during busy times, such as screen scrolls.

A tab list follows, running from \$1D80-1DFF. This list allows 64 different tabs to be set at any one time. Because tabs well into a paragraph are allowed, two full bytes are reserved for each tab value.

The four WPL \$A through \$D strings are stashed next in our low memory work file area. Each can be up to 64 characters long.

The area from \$1F00-1FFF has several uses. Sharing is possible because each use is temporary. The entire buffer formats the top or bottom line in its expanded form with real page numbers and full space padding. Other uses split this page in four temporary work files, starting with a short multiple use buffer at \$1F00-\$1F3F. This buffer is used to assemble PRT and TAB filenames, to hold slot and drive values, and to substitute for the normal WPL work buffer.

An = filename buffer follows from \$1F40-1F7F. Hold here is the old text filename used while ProDOS is being temporarily diverted for something else. For instance, the name of a glossary or a WPL file to be loaded may need the active filename buffer at \$B700. During such special use, the old filename is briefly saved here. Immediately after special use, the old filename is returned to the active buffer at \$B700, then is followed by an = buffer at \$1F80-1FBF, used for repeat searches and replaces.

Next comes an [open apple] or [closed apple] stash at \$1FC0-1FFF, used as the second half of the type-ahead buffer.

Let's next skip way up in memory to . . .

## ***The High Work File Area***

When the upgrade to ProDOS Applewriter 2.0 was made, another work area was needed. Because any ProDOS application or .SYS program must always start at \$2000, the most sensible thing to do was put the main code at \$2000 and then cram the rest of the work files as high in main RAM as possible.

The high work file area runs from \$B600-BFFF, as shown in the memory map of Figure 7.3B and detailed Listing C3. The tab status image runs from \$B600-B6F0. You need all that room because you have a possible screen width of 240 columns. The tab image is simply an ASCII snapshot of the tab status line. Unset tabs are shown in normal text. Set tabs are shown in inverse. The higher "fives" markers get progressively longer. This way, you can tell where you are on a very wide screen line. An apostrophe (') is



thus used for 5-95, an exclamation mark (!) for 105-195, and a bar (|) for 205-235. The tab status line doubles as a display column counter.

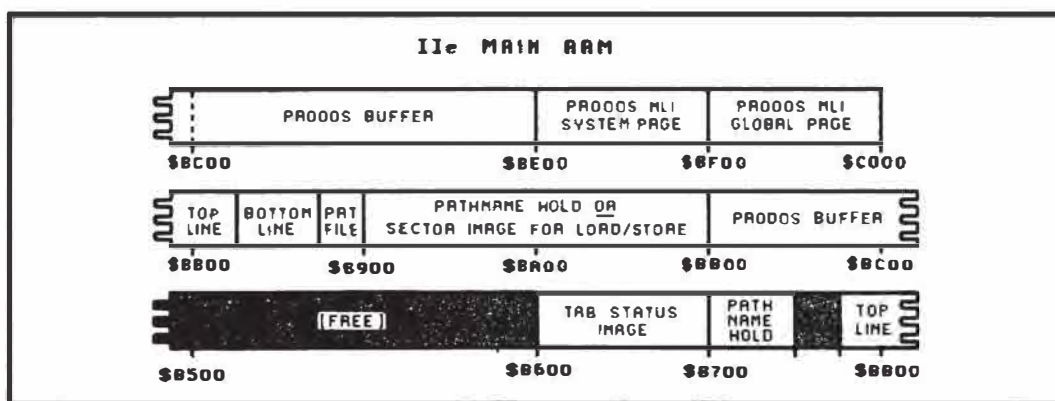


Fig. 7.3B. Simplified memory map of Appewriter 2.0 high memory work files.

A composite path name for ProDOS is usually accessed from \$B700-B73F. Although a ProDOS filename is only allowed to be 16 characters long, the combined prefix, subdirectory, sub-subdirectory, and filename can go to a maximum of 64 characters.

The print/program file values are next in line and take up the area from \$B7C0 through \$B8FF. Provided first are two 128 character buffers for the top line and the bottom line. The top and bottom line, with delimiters and a # for possible page numbers, are held here in their compact form. When formatted, the top line or the bottom line is expanded as needed to a multiple use buffer way down at \$1F00 into its stretched or open form.

These compact line buffers are followed by individual stashes of all the print values, such as the page number, the justification mode, and so on. Locations of the individual print/program values are detailed in Listing C.3. Our intent here is to get a look at the big picture. More detailed listings will follow shortly.

A 512 byte ProDOS user buffer at \$B900-BAFF follows the print value stashes. The user buffer can hold one sector for loads or stores that involve fancy delimiters or can be used simply as a pathname hold.

A second buffer follows from \$BB00-BDFF. These 768 bytes are needed for internal ProDOS uses and are specified by various ProDOS links throughout the program. The area from \$BE00-BEFF is reserved as a ProDOS systems page, as might be needed by BASICS.SYS or whatever. Although this area is not used, it is carefully preserved in this program so that applications can be transferred in an orderly manner.

Finally, the \$BF00-BFFF page is the ProDOS globals page, and holds information needed to access ProDOS. In particular, a JSR \$BF00 will start

some ProDOS action, which is set by a short file following the calling code. More on this shortly. Other important locations on the globals page also are shown in Listing C.3.

As you can see, bunches of work files are needed by ProDOS Applewriter 2.0. These work files hold things likely to change during your use of the program. Much of the power and uniqueness of Applewriter comes about through extensive and creative use of these work files.

Our survey of the work files is complete. Next, let's briefly look at . . .

## ***Internal File Area***

The internal file area holds stashes, commands, and file values that are stuffed into the working code. The majority of these provide the data files needed for ProDOS access by way of the MLI machine language interface. Others hold local variables or provide convenient stashes. Listing C.4 shows all of the internal work files.

We'll save details on the ProDOS internal files for just a bit. Other internal files include local stashes, a filler and emptier for the modem type-ahead buffer, a BASH table of screen base addresses, a glossary nest pointer that you use to call glossary entries eight deep, path name parameters, local X and Y register stashes, a serial transmission delay table, a list of baud rates, and a stash for catalog attributes.

The use of each internal work file will become fairly obvious when you study the related code module. The important thing about internal work files is that you must exactly bypass them if you are capturing source code. Otherwise, you will get aliasing and starting off on the wrong foot problems, and your capture attempt may fail outright.

The final file area we are interested in is . . .

## ***Reference File Area***

The reference files hold things that are more or less permanent and that do not normally change much during program use. The reference files are stashed in main RAM from \$5475-5FFF. Listing C.5 provides details on these reference files.

The first reference file is both a menu and a prompting list of available functions. There are 32 possible control command functions. The 27 that are actually used range from [ @ ], which is really [ delete ], up through [ \_ ], the page/position display toggle.

Some of the control commands are missing and some are hidden as dedicated keys.

Here is a summary of the . . .

### "Funny" Control Commands

---

1. *Dedicated use*
  - [@] — [delete]
  - [H] — ←
  - [I] — [tab]
  - [J] — ↓
  - [K] — ↑
  - [U] — →
  - [\_] — [page/position]
2. *Not available for use*
  - [M] — [carriage return]
  - [ ] — [esc]
3. *Not used*
  - [i] — \$1C (FS form separator)
  - [ ] — \$1D (GS group separator)
  - [!] — \$1E (RS range separator)

[delete] is temporarily recoded as an \$80 because its \$FF value is reserved as an internal text file marker. It is stripped before it is allowed to enter the text file. As a reminder, \$00, \$7F, \$80, and \$FF markers are reserved exclusively for internal Applewriter use and must never be allowed in a text file.

The NULL command may be needed by many older printers as part of an imbedded command. To allow NULLs, you use a [\_] in your text file. This is really a user separator. The print routine then automatically changes *all* user separators to NULLs, like it or not. We saw ways around this earlier.

The arrow keys are really the control functions [H], [J], [K], and [U]. [I] is the tab key that moves the cursor to the tab settings. If you want an escape *without* pressing [esc], you could alternately use [ ] instead.

The function list is used two ways. It is first scanned for a match to a control key command. If a match is found, that specified action is done. If a match is not found, nothing happens.

Secondly, the function list is scanned for those activities that need a user prompt. For example, on a save, The [S]ave gets copies to screen.

A list of function addresses follows the function list. These routines are detailed in Listing C.4. When a match is found between a control key and the function list, a jump is made to that function address. This carries out the requested action.

Note that these addresses are *one less* than the start of each function. This is because these addresses are accessed by way of the "forced subroutine return" method of option picking. This method always goes on the address pair shoved onto the stack *plus one*.



Next in the reference file area is a pair of back-to-back matching files. The first of these files holds two letter pairs of print constants, and the second holds two letter WPL commands.

On a print constants match, the value is gotten, converted to hexadecimal, and stored as needed in the print constants work file. On a WPL command, the needed action is carried out. A list of WPL addresses follows the WPL command list. As before, on a two letter match, that action is done. A peek ahead to Listing C.7 will show you which action goes where.

The reference file continues with some ASCII coded prompts. These prompts handle such things as page and line displays, modem prompts, filename prompts, and the .PRT and .TAB trailers. By the way, these trailers are used as postfixes in ProDOS Appewriter 2.0. In Appewriter IIe, they were used as prefixes. Note that neither the print constants nor the tab files are interchangeable between the two program versions.

Next is a list of ProDOS error messages. Each message begins with a ProDOS error number followed by the actual error message in low ASCII. Following this list are three more ASCII error prompting stashes for ProDOS, [F]ind, and WPL.

At \$5897, you will find a list of WPL error messages, presented in conventional low ASCII. The error message needed is found by counting the number of \$0D carriage returns required to reach it.

Following this location is a three-byte "SCP" stash used for tab set, clear, or purge. The ProDOS command menu follows the three byte stash. This menu is produced by [O] and lists the ProDOS options onscreen. We already found out how you can *shorten* this menu to make room for other patches and leave more catalog information onscreen.

As before, the addresses follow the prompts, with the ProDOS access addresses being listed from \$5A14-5A27. Once again, these are the entry points *minus one* as needed by the forced subroutine jump method of option picking.

Some random stashes follow the command menu. This includes [A]djust prompts, [O] catalog space on disk messages, and the return prompt. This location is followed by a list of ProDOS file types, shortened to *six letter mnemonics*.

After this list come some more low ASCII messages for the catalog display, a help pathname, and the Y, N, and A needed during [F]ind and replace.

All of which brings us to \$5C43 and the startup screen, which holds the first screen image you see on bootup. Following this location are the additional functions menu and the additional functions addresses. These addresses are also *one less* than the routine they point to, as needed by the forced subroutine return option picking method.

A four byte print value stash follows. This dude is used to hold the [W] and [X] stop characters, followed by the left and right margins for the top and bottom lines.

Wrapping up our reference files is the fixed portion of the print/program screen display.

The bytes from \$5FF0-5FFF are unused in the original program but are loaded with the program. Several of our earlier patches need both this area and then some. This patched version continues on to \$6020. With your own patches, you are free to go on up to \$B5FF, a total of 21983 extra bytes. Lots of room is available for improved goodies of your own.

As we've just seen, the reference files hold things that seldom if ever change during program use. The reference files are loaded into the machine, following the main word processing code. Remember that the reference files and internal files are loaded off disk. Most of the work files are created, initialized, and then used by the running program.

The next big question is . . .

## ***How Do You Crack Page Zero?***

Many of the great mysteries to be solved by the tearing method involve strange and wondrous uses of page zero. Page zero addresses on any 6502 system are very handy because they are easy to access. More importantly, certain address pairs that let you do 16 bit "anywhere in memory" access positively *must* sit on page zero. If you do not thoroughly understand what is on page zero, how that information gets there, and how the information is used, you cannot possibly understand *any* Apple program.

Important uses of page zero include *pointers*, *counters*, *stashes*, and *flags*.

### **Important Uses of Page Zero**

---

1. *Pointers*

A pointer holds an address or an address pair that finds some other memory location.

2. *Counters*

A counter remembers positions or trips needed, which often start at some value and are decremented to zero.

3. *Stashes*

A stash temporarily holds some value that is likely to be changed.

4. *Flags*

Flags remember conditions and operating modes. A flag often has only two or three possible values.

The big advantage of using page zero for pointers, counters, stashes, and flags is that you can reach page zero locations faster and with fewer bytes of code.

Pointers hold addresses. A single pointer can address only 256 different places in memory. A dual pointer can address 65,536 places in memory. Dual pointers are most often used with the 6502's very powerful indirect indexed addressing mode. Indirect indexed addressing lets the computer reach anywhere in memory without worrying about page breaks or 256 byte limits. For instance, in Applewriter, a LOCURS pointer points to the current cursor position in the LOFILE half of the text file.

By the way, if you are rusty on addressing modes and machine code in general, check Don Lancaster's *Micro Cookbooks* (SAMS #21829-21830). Volume II on machine language programming should be of special interest to you.

Counters hold something that is being incremented or decremented until some magic value occurs. Most often, a counter is initialized to its maximum value and then decremented to zero. This initialization is done because zero is testable free with the BNE command. In Applewriter, there's a deletion counter that makes sure that you do not delete words or paragraphs which are longer than 1024 characters.

Stashes hold values that may or may not change. In Applewriter, a stash is available that remembers the length of filenames.

Flags remember conditions for you. Usually, a flag will only have a few possible values, with a "don't" value of \$00 and a "do" value of \$FF being common. In Applewriter, an R flag remembers whether you are in the replace mode.

The big question now is "How do you find out what the flags do?" Answering this question is one route to . . .

### Cracking Page Zero Locations

---

1. Create a notebook of all used locations as listed on the cross reference for the study program.
2. Tear the program apart, putting into the notebook all known information about flag uses.
3. Go back to the cross reference area and color code each page zero use—red for writes and green for reads. Correct any errors or omissions in the notebook.
4. Write a complete script of page zero use. Use both a summary list and a detailed script.



As with most other things in the tearing method, a few passes are required to get it right. On the first pass, you find out whether a particular location is used. During the second pass, you find out roughly who uses that location for what. In the final pass, you nail down the exact use details. I like to take a small notebook and put two or three page zero addresses on each page. You get these addresses from your page zero cross reference listings.

As you go through the tearing method, some page zero uses will leap out at you. For instance, in Applewriter, the [V] command vectors to \$35B6, which changes the state of the \$72 flag from \$00 to \$FF or vice versa. Obviously \$72 is our [V]erbatim flag, and we are home free on this one.

Record everything of interest you find in the notebook. Even if you don't have the foggiest what the location is up to, knowing that [L]oad and [S]ave need that location and that it gets initied to some value will help later.

The notebook should completely crack about one-third of the page zero locations. Another third should be pretty nigh but not plumb. And the final third of these locations will still have some mystery surrounding them. Regardless of how far you get, complete the main tearing process as far as you possibly can.

Pay particular attention to whether a page zero location is global or local . . .

1. *Global Value*

Something that has only one possible meaning or use during the entire program.

2. *Local Value*

Something that can have many different meanings at different times and at different points in the program.

Global values obviously are simpler to handle. If you find the same location first being used by two or more wildly different portions of the code, assume that location is used locally. Then prove yourself right or wrong.

As contrasting Applewriter examples, the [R] flag at \$F5 is used everywhere in the program to pick *inserting* versus *replacing*. The Y-register stash at \$C5 is used many different places in the program for many independent things.

Note that the second page zero location in a pointer pair often goes along free for the ride. Thus, you might initialize both \$80 low and \$81 high as a pointer pair, but you would only refer to the \$80 in a *LDA (80), Y* command. The use of \$81 as a "page" or "high address" is inferred in the

*LDA (80), Y* command. This inference is true for most double wide pointers.

At this point, your page zero should be around half cracked. Less than a third white margin should remain in your main tearing attack.

Next, go back to your cross reference listing and start color coding each page zero use. Color one location at a time with red for writes and green for reads. As you color each location, update the notebook with who uses what how. Pay particular attention to what *sets up* that location, who *changes* that location, and which code *tests* that location. This should make the use of each flag obvious.

Correct and expand your notebook comments as you crack each location. Remember that any booting or cold start code can initialize certain locations to certain values. The booting code in AW.SYSTEM sets a ProDOS file buffer to \$BB00 in main RAM. The cold start code in AWD.SYS sets at zero all page zero locations from \$60 to \$FF.

If you have an all green read-only location, always go back to the cold start code to see what got stashed where. If some location seems always to be stuck in a certain value, make sure no branching, indexing, or indirect storing changes this location in a subtle way. Find out also whether different or older versions of the program might have needed this location for obsolete uses.

At any rate, the way you crack page zero is to first record the obvious and make some guesses. Then go back and study each page zero location one on one.

## ***Applewriter Page Zero Uses***

Listing C.5 gives you a summary of the page zero uses of Applewriter, broken down into counters, flags, pointers, and stashes. Each listing is shown in order of increasing address. More detail appears in Listing C.6, which gives you a complete script of all the page zero uses.

I guess a program counter really is a pointer, so some overlap occurs between the pointers and the counters. And the line between a multivalue flag and a few-valued stash also gets a tad thin at times.

Probably the single most important page zero locations are \$84 and \$85, the LOCURS pointer pair. This pointer pair points to the address in the text file in auxiliary memory where the next character is to be entered or removed.

As is customary in the 6502 world, the "low", or "position" address appears first in \$80 and the "high" or "page" address is stashed last in \$81. For example, if \$80 holds a \$46, \$81 holds a \$35, and the Y register holds a \$00, the command *LDA (80), Y* goes to address \$3546. Then the command copies what it finds in the accumulator to that address. Should the Y register have been holding an \$02 instead, the same command would go to address \$3548. Note that this command can reach any location in the entire address space just by changing the values in \$80, \$81, and the Y register.



Other double wide pointers access HIFILE to print characters in order, to display to the screen, and for various other uses that need to access bunches of characters in sequence.

Be sure you understand exactly how indirect indexing works. Understanding LOCURS is first and foremost in this quest.

At this point, you should now be halfway through understanding this program.

We now know how the text file area works. We have studied uses of the work file area, the internal file area, the reference file area, and page zero.

Next on the agenda are the . . .

## ***Entry Points***

Entry points are those locations in the code where you go to do something . . .

### **Entry Point**

---

Some location in a block of code that you go to start something happening.

You can use several possible entry levels, depending on what you need to get done . . .

### **Entry Levels**

---

1. *High Level*  
Points entered in the whole code by another system to run, rerun, or process errors.
2. *Command Level*  
Points entered in response to a main menu selection.
3. *Module Level*  
Points entered to handle specific tasks or submenu selections.
4. *Service Level*  
Important subroutines that do all major housekeeping and handle any often needed utility functions.

Listing C.7 summarizes the important entry points. A complete and detailed disassembly script appears in Listing C.8. This one will tell you more than you could possibly want to know about every module in the working code.

There's no single answer to the obvious question of "How does Applewriter work?" How you answer depends on what you think is important and where your interests lie. And any attempt to go through the code in numeric order is pretty much fruitless because you lose track of who is doing what to whom.

Instead, let's see whether we can't thread together some of the important working concepts of this program. Our first concern should be the . . .

## *ProDOS MLI Links*

The ProDOS used in ProDOS Applewriter 2.0 is totally stock in every way. The program is also installed in its normal space in high main RAM. All ProDOS access is by way of its MLI, short for *machine language interface*, because klutzy and RAM gobbling BASICS.SYS is not used.

To understand ProDOS, you will need Apple's *ProDOS Technical Reference Manual* and Quality Software's *Beneath Apple ProDOS*.

Let's see whether we can't give you a few hints. The usual LOAD, STORE, OPEN, etc. commands do not exist when using the MLI. Each time you want to access ProDOS, you do a machine language JSR \$BF00, immediately followed by a three byte *data file*. The first byte gives you the command, and the second two bytes point to a second data file needed to complete the command. The complexity of the file pointed to varies with the command. This second file typically involves less than a dozen bytes. Everything—repeat, everything—goes to or comes from ProDOS by way of the JSR \$BF00 MLI interface.

On the facing page is a ProDOS command summary . . .

More details appear in the various listings and in your tearing of the code itself.

The ProDOS interface is far more uniform and far more flexible than was DOS 3.3e. For instance, the exact same command saves a text file, a BASIC program, or a binary file. The only difference lies in the attributes of the file at the time it is written.

The tab and print constant files are loaded and saved as binary images, putting each in its respective slot in the work files.

Glossary and WPL program files are treated similarly, except that they are text files and as before, are loaded into specific places in memory. In fact, the WPL loader does double duty as a glossary loader. The loader is simply tricked into putting what it reads in the wrong place when loading a glossary. These files are all read to or from main memory.

### ProDOS MLI Access Commands

---

- \* \$40 - Allocate interrupt
- \* \$41 - Deallocate interrupt
- \$65 - Quit
- \* \$80 - Read block
- \* \$81 - Write block
- \* \$B2 - Get time
- \$C0 - Create
- \$C1 - Destroy
- \$C2 - Rename
- \$C3 - Set file info
- \$C4 - Get file info
- \$C5 - Volumes on line
- \$C6 - Set prefix
- \$C7 - Get prefix
- \$C8 - Open
- \* \$C9 - Newline
- \$CA - Read
- \$CB - Write
- \$CC - Close
- \* \$CD - Flush
- \$CE - Set mark\$
- \$CF - Get mark
- \$D0 - Set end of file
- \$D1 - Get end of file
- \* \$D2 - Set buffer
- \* \$D3 - Get buffer
- \* - Not used by AWD.SYS

These are also total reads or saves, in which the entire file is loaded or saved at once. Your text files may or may not want to use a total load or a total save.

Applewriter text files often get moved one 512 byte sector at a time onto or off of the disk. This gives an orderly way to search for the delimiters that allow partial loads and saves. Moving one sector at a time also solves a memory management hassle because ProDOS will normally load or store into a buffer in main RAM. After searching or processing, the needed pieces of the loaded or stored text are transferred to auxiliary RAM by the memory management code.

Random access a sector at a time is done with the SET.MARK and READ.MARK commands. Appending is done similarly with the SET.EOF and READ.EOF commands. Generally, a file must be created, opened, read or written to, and finally closed.

A catalog display is done using GET.FILE.INFO. This is handled by routines internal to Applewriter. Files are locked or unlocked by changing file attributes and then using SET.FILE.INFO. Files are deleted with DESTROY.

Several [O] options are unique to ProDOS. Each ProDOS disk *must* have a prefix. Unlike the volume name everyone ignored in DOS 3.3, this prefix must be remembered and available at all times. You also cannot change a disk in a drive without also changing the prefix. A SET.PREFIX command exists. As we saw in Chapter 6, a one key glossary entry can greatly ease prefix setting hassles.

The [O]-F option is used to list the prefixes of the volumes on line. Because ProDOS has no init code, the volume formatter gets a separate program called FORMATTER and installs it from \$0800-17FF in main RAM. This code module is then run, doing an init for you. The same module also completely and destructively overwrites the glossary, your WPL file, and any footnotes in use.

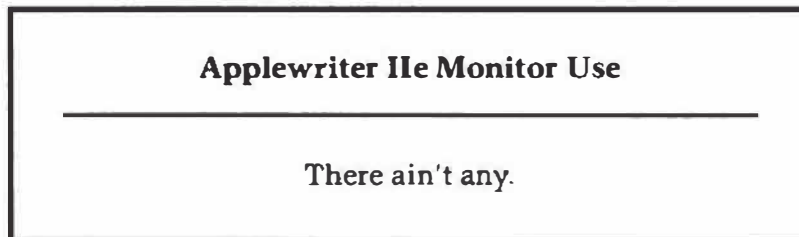
You use the final [O]-J option to set IIC modem or printer parameters. What happens is that the baud rate, start and stop bits, parity, etc., are coded in a proper form to set the 6551 serial interface chip in the IIC. The *IIC Technical Manual* gives full details.

Unlike earlier versions of this code, you should have no problems when installing ProDOS Applewriter 2.0 onto virtually any hard disk. This happens because the operating system used is totally standard and the program is completely unlocked and movable.

'Nuff said on ProDOS. Let's go on to . . .

## Monitor Access

ProDOS Applewriter 2.0 monitor use is nonexistent.



ProDOS Applewriter 2.0 uses zero—repeat, zero—monitor routines. The ROM is never switched into the high memory area. All of the key getting, disk accessing, and character outputting is done internally to Applewriter. The good news here is the total control you get with a built in,



type-ahead buffer, screen lines up to 240 characters, controllable scrolling both vertically and horizontally, ease of output routing, and so on.

The bad news is that some parallel cards on the IIe expect normal use of normal monitor routines. In particular, the IIe does not use location \$24, in which many parallel printer cards *demand* to find some horizontal cursor position information. As we have seen, custom patches are needed to handle these problem cards with version 2.0. A partial fix is available with the 2.1 update.

Add-on video cards usually will not properly access Applewriter because all screen output characters are handled internally. In fact, these cards are all carefully disconnected by Applewriter as part of the cold startup process in AWD.SYS.

Actually, with horizontal scrolling to 240 characters, nothing special is needed in the way of character display. To further prevent tampering with the internal Applewriter routines, the usual keyboard input hook KSWH and KSWL (\$38 and \$39) are set to point to a "brick wall" RTS and then are studiously ignored.

The CSWH and CSWL character output hooks (\$36 and \$37) are not forgotten. Instead, their primary and only use is to let a serial or parallel interface card adjust them slightly for proper printing. Ferinstance, if you set this printer hook to \$C100 and send a \$00 NULL to the interface, the interface will usually reset the hook to \$C105 or something similar. ProDOS Applewriter 2.0 uses this hook only to get the interface card started and set to the right I/O address. The card then grabs the corrected address for its own internal use.

Characters are output by an internal and protected version of the usual \$FDF0 (Fideyfoo) COUT hook. This output is handled by a pointer pair at \$9E and \$9F on page zero that handles the internal COUT destination setting.

Applewriter's internal COUT can point several possible places . . .

#### **Applewriter Internal COUT (\$9E,9F)**

1. *On a .pd0 print to screen*  
Points to \$4415 screen code.
2. *On a .pd1 or .pd2*  
Points to I/O space as adjusted by the interface card or circuit.
3. *On a .pd8 print to disk*  
Points to \$4397 disk write code.

## Memory Management

As we have seen, the text files are in auxiliary memory, and everything else is in main memory. Text files are accessed by some code down on page one that does not change as the memory is switched between main RAM and auxiliary RAM. Routines in main RAM that need to read the text file do so via these page one access links.

Remember that the auxiliary RAM text file is really two files. LOFILE starts at \$0801 and builds up, and HIFILE starts at \$BDFE and builds down. \$FF markers define the beginning of LOFILE and the end of HIFILE. The open ends of both files face each other across all the remaining empty space. These open ends are identified with \$00 markers. LOFILE holds everything from the start of the message up to one less than the current cursor position. HIFILE holds everything from cursed character to the last character in the file. Characters are normally entered into the top of LOFILE. All of the characters are entered as low ASCII, but another routine carefully re-marks each end of each screen line with a high ASCII character instead.

Most of the usual routines enter things to the top of LOFILE. Others will pass a character from LOFILE to HIFILE to back up the cursor. Yet others will pass a character from HIFILE to LOFILE to move the cursor forward. [B] and [E] are extreme examples.

Several pointers access the text file. These pointers include LOCURS and HICURS, which point to the open ends of LOFILE and HIFILE. You will also find a screen pointer that starts at a point in LOFILE equal to the top screen line, advances through LOFILE to the cursor, then automatically switches to HIFILE to continue. The screen pointer keys on high ASCII characters to count screen lines.

A printer pointer is used to scan through LOFILE to get characters. Because everything is moved to LOFILE before printing, no switch to HIFILE is needed by this pointer. A general use pointer pair accesses either LOFILE or HIFILE as needed.

Another specialized pointer (\$AE,AF) will back up automatically to the first high ASCII character that this pointer finds. This character locates the start of any screen line and is useful for both screen formatting and tabbing.

These pointers all work by switching to read auxiliary RAM, getting a value, then immediately switching back to read main RAM. Certain other routines will write to auxiliary RAM by switching to it, doing a store, then switching back to main RAM.

Note that the writing routines can be in main RAM without a conflict. Only the reading routines *must* be in a portion of the memory that is not switched between main and auxiliary RAM. Otherwise, as soon as the main-auxiliary switch is flipped, the op codes being read vanish.

As we have seen, no switching into the monitor ROM even takes place. Nor is auxiliary page zero or auxiliary high RAM ever activated.

## Character Entry

No use is made of the monitor KEYIN routine. If you tried using KEYIN with a word processor, you probably would drop keystrokes during hectic typing times. Instead, ProDOS Applewriter 2.0 uses its own internal routine to get keystrokes. This routine includes a 64 key, type-ahead buffer. If your typing gets ahead of the processing, up to 64 keystrokes are saved in a pair of storage buffers.

The main keystrokes are saved to the character buffer at \$1D40, and [open apple] and [closed apple] keystrokes are separately saved to the apple buffer at \$1FC0-\$1FFF. Remember that the apple keys as well as the main keystroke must be saved, or the computer would not handle certain functions correctly. Two round-and-round pointers keep track of where you are in the key buffer. A filling pointer \$F3 and an emptying pointer \$F2 take care of this task.

During non-hectic times, the filler and the emptier stay together, and the keystrokes are immediately used. At other times, the filler gets ahead, and characters are saved to the buffer. Routines that take lots of time automatically check the keyboard every now and then to make sure nothing gets missed.

A busy signal (\*) prompt appears on the normal status display during busy times.

As we saw a while back, characters can still get missed every now and then if a sloppy typist, a bug in the keyboard encoder, and the slower insertion mode all gang up on the key buffer. The buffer seems to be working perfectly when characters are lost. The buffer access is what fouls up the works.

Reviewing, characters can be gotten directly from the keyboard during non-hectic times and otherwise gotten out of the type ahead buffers when things happen too fast.

Several other character sources exist, in addition to the user. Down on page zero is a special WPL and glossary activity flag \$DF. Bit #7 or the MSB N slot of this flag controls WPL activity, and Bit #6 or the V slot controls glossary activity. If the glossary is active, the character is gotten from the glossary file. Similarly, if WPL is active, the character is gotten from the WPL program file. Sometimes the WPL file will involve itself with its \$A-\$D strings. If WPL and these strings are active, the \$A-\$D string becomes the source for the next character to be used. You'll find a separate string activity flag at \$F6 to handle \$A-\$D activity.

Sometimes you want to use a string already in the machine, such as the = filename or something else that has been previously formatted or put together. A special controlling string flag \$AD exists for such cases. If this



string is set, the old string, which is usually in the key buffer at \$0200, is used one character at a time. If the string flag is cleared, new characters are gotten as needed from the user, the type-ahead buffer, the glossary, WPL, or the \$A-D flags.

Yet another source for strings of characters exists. When doing a [Q]-I, you can receive characters directly from a modem or by way of a modem buffer that saves incoming characters during hectic times. This access bypasses the usual key-getting routines.

ProDOS activities, such as loads and stores, completely bypass any key-getting routines and usually put their values directly where they belong. If searching for delimiters is needed, it is done one 512 byte sector at a time by way of a user buffer at \$B900.

The majority of the word processor's time consists of patiently waiting for the user to input a new keystroke. Regardless of a keystroke's source, after that keystroke is received, it is filtered for control and cursor motion commands. If a valid command is found, it is carried out. If not, the character is entered to the top of LOFILE.

Summarizing . . .

#### Sources of Keystrokes

---

1. Directly from the user during non-hectic times.
2. Indirectly from the user via a type-ahead buffer when the processor gets busy.
3. From the glossary during glossary activity.
4. From the WPL program during active use of WPL.
5. From the \$A-\$D strings if these WPL strings are active.
6. From an old string already in the keybuffer if that string is still needed.

We have seen that several sources of keystrokes are available, all of which are handled internally by the code. User input is accepted directly or is stashed in a pair of buffers if the processor is busy.

Characters can also come from the glossary, from WPL, or from a WPL \$A-\$D string if the controlling flags are set properly. Sometimes, an old string will be reused instead of getting new input. And finally, characters can come directly from the modem or by way of the modem's type-ahead buffer, bypassing the usual key getting routines.

Now for some details on the . . .

## Screen Display

The screen display has some very sneaky and complicated code associated with it. First note that you can turn the screen off and on with flag \$F7. Leaving the screen off speeds up WPL's operation considerably.

Naturally, seeing what you are doing when the screen is off is tricky. A screen that is turned off is useful, though, to display WPL menus, prompts, and whatever.

Before a screen display is updated, any routine that messes with the text files will automatically reformat the screen lines in that file. Reformatting is done by backing up two lines from the cursed position and then counting how many whole words will fit on a line. Each line stops either on a carriage return or when the line does not have enough room for the next word. At that point, a marker character, usually an \$0D carriage return or an \$20 space, is changed to a high ASCII \$8D or \$A0 and restored to the text file. All old low ASCII characters are erased from the text file. The process continues forward through the text file until a carriage return is found that is already correctly formatted.

Note that anything two lines before the current activity had to be correct already, thanks to previous reformatting. Everything beyond the next carriage return is also correct. Only the mess in the middle needs straightening out. The *entire* text file is reformatted after a margin altering [A], after loading, after printing, and any other time that something really major happens.

Completely reformatting a long text file may take several seconds. The upshot is that, before a screen update, all of LOFILE and all of HIFILE have end of screen line markers properly placed to end each line on a whole word.

The cursor usually stays on the middle line of the active screen. Should the screen overflow, everything scrolls up one line. Should it underflow, everything backs down one line. During insertions, characters get turnstiled as far as they have to in order to reach the next carriage return.

To update the full screen, the screen pointer pair \$88,\$89 backs up 12 lines, which is usually 12 inverse-ASCII characters from the top of LOFILE. Characters are removed from LOFILE and put on the screen up to the cursed location. Immediately beyond LOCURS, the pointer is moved to HICURS, and the code continues filling in characters from HIFILE until 12 more lines are completed.

The flashing you see on the cursed character is purely your imagination at work. The service routine that awaits a keystroke patiently flips the cursed character on the screen between low and high ASCII. Sometimes that character is left as an inverse low-ASCII marker. An example is the cursor on the nonactive side of the split screen.

Note that the large and empty no man's land between LOCURS and HICURS is bypassed. The lowest character in HIFILE ends up at the cursed location. Note also that the alternate character set is used, which

has no flashing characters available. Low ASCII characters appear as inverse text.

Only the active half of the screen is updated on a split screen. The inactive half of the screen reviews static, remembering things the way they were.

If the wraparound flag \$E1 is not active, characters are put on the screen wall to wall without regard for word breaks. Only a 79 character line is used because room must be left for the optional carriage return display to appear in column 80.

A user prompt is sometimes needed at the bottom of the active screen. To print a prompt on the screen, three lines are erased, and the prompt is placed on the middle line. Prompts are normally read as needed out of the reference file area. Service subs are built into the screen code for the live cursor screen motions, line clearing, scrolling, and so on.

Another summary . . .

### Screen Updates

1. Before any screen update, low ASCII markers are placed at the end of each screen line in the text file.
2. Everything before the cursor on the screen comes from LOFILE.
3. The cursed character and everything beyond comes from HIFILE.

Things get more complicated if you are using a right margin wider than 78 columns.

In this case, not all of the screen line can be displayed. A special stash is used to calculate the offset needed between the previous end of screen line marker and the actual screen starting text character. This offset is automatically added when finding text file characters to go on the screen.

As long as the cursor stays near the middle of the screen, no change is made in this offset value. If the cursor gets left of the twelfth character, the offset is decremented, giving an apparent horizontal scrolling of one character to the left. Should the cursor get right of the sixty-eighth character, the offset is incremented, giving you an apparent horizontal scrolling of one character to the right.

The display will start with its left margin LM value in column zero, which produces an apparent what-you-see-is-what-you-get display, as long as the screen width is less than 78 characters total. For the most exact display, use [tab] rather than PM values for your paragraphs. Note that all



screen lines will be justified flush left even if a wide left margin is used. Note also that breaks on whole words only are required on lines wider than 78 characters.

We now know something about how ProDOS works, how the monitor is used, where the characters come from, how they are managed, and how the screen update works.

Next are two . . .

## *Individual Control Commands*

Let's run down the control command list, seeing roughly what each command does. For more detail, check Listing C.8 or your own torn disassembly listing and cross reference list.

[@] is really [delete], recoded to \$80 from its default value of \$FF. This command unconditionally knocks out LOFILE's uppermost character and replaces it with a \$00 marker. The command then backs LOCURS up one character.

[A] is the command to alter the screen margins. If the characters per line are less than 78, the left screen margin is set to appear in column zero. If the characters per line are more than 78, the left screen line is first set to center the cursor if possible. As the cursor gets moved within 12 characters of either the left or right margin, horizontal scrolling is activated. Screen lines are marked by setting the last character in each screen line to high ASCII.

[B] moves all the characters from LOFILE to HIFILE, placing the cursor at the beginning of the text. When finished, LOFILE will be completely empty, and HIFILE will hold the text being processed.

[C] changes the case flag, initially from none to U or later from U to L or from L to U. When characters are entered, this flag is checked. If active, uppercase or lowercase is forced as chosen. The flag is reset on all cursor motions except the left and right arrows. These arrows let you capitalize or lowercase as many characters in a row as you want. Only real letters are changed.

[D] toggles the data direction flag between < and >. If a [W] or [X] is specified with a data direction of >, words or paragraphs are *restored*. If < is the data direction when [W] or [X] are specified, words or paragraphs are *deleted*. The data direction flag also sets the direction of a search or search and replace.

[E] moves all the characters from HIFILE to LOFILE, placing the cursor at the end of the text. When completed, HIFILE is completely empty, and LOFILE holds all of the text being processed.

[F] does either a search or a search and replace. Delimiters are interpreted, substituting special ones if used. Then the text is searched using the \$98,99 pointer pair. If you want to make a replacement, text is moved from HIFILE to a work buffer and the replacement is made. Various

options substitute for fake carriage returns, allow repeats for all occurrences, let you use wild cards, and provide any length capabilities.

[G] either sets up or reads the glossary. If a valid read, the glossary flag is set. If set, characters are gotten from the glossary work file until the next carriage return. At that time, the glossary flag is cleared. If the flag is a \*, the glossary is emptied by placing a zero at the glossary start location \$1B00. If the flag is a ?, the end of the glossary is found and the new definition is entered that ends with a carriage return and a \$00. The glossary has a *nest* that works like a subroutine and remembers up to eight return pointers. This nesting picks back up on the caller when the callee is finished.

[H] is the left arrow. When it is the only key pressed, it backs up one location by moving one character from LOFILE to HIFILE. When used with [closed apple], the left arrow ([H]) does an express by-word backspace, continually backing up until the first space is found. With [open apple], the left arrow saves a character to the swallow buffer instead of HIFILE and increments the round and round swallow buffer pointer \$AC.

[I] moves the cursor to a tab. The present position since the last carriage return is calculated. A test is then made to see whether any valid tabs exist beyond the present position. If so, spaces are added to the top of LOFILE to move to the next tab position. If [closed apple] happens to be down, the cursor is moved without space padding so that the characters are tabbed over without being moved. Tabs are permitted anywhere within a paragraph.

[J] is the down arrow. When it is the only key pressed, it moves characters from HIFILE to LOFILE, repeatedly frontspacing until one line is moved. Each succeeding line ends with a high ASCII marker. With [closed apple] and if enough text is left, the down arrow goes forward 12 whole lines.

[K] is the up arrow. It *moves* characters from LOFILE to HIFILE, repeatedly backspacing until one line is moved. Each preceding line ends with a high ASCII marker. With [closed apple], the up arrow tries to go backward 12 whole lines if enough text is available.

[L] is the load command. Loading can be from the text file, which is really a copy command, or from ProDOS. Loading from ProDOS is first done via a one sector, 512 byte buffer at \$B900 in main RAM. After scanning for any needed delimiters, the characters are transferred to the top of the LOFILE text file area in auxiliary RAM. Text is entered just beyond the present screen position. Alternate delimiters provide for all occurrences, wild cards, and fake carriage returns. An option exists to load only to screen.

[M] is the carriage return that ends each command. This command is not available for other uses, although you can fake a glossary carriage return with a ] and a search for a carriage return with a special delimiter, such as >.

[N] is the new command. Because this command can be deadly, you are given a prompt that needs a Y answer. If you are serious about destroying your text file, this command adjusts the HIFILE and LOFILE pointers so that nothing is in either HIFILE or LOFILE and your cursor is sitting at the beginning of LOFILE. The old material is not erased, except for the first character. All that happens is that the first character gets replaced with an open-end-of-file \$00 marker.

[O] is the DOS access menu. The menu is displayed and a selection is gotten. On a catalog command, a GET.FILE.INFO is done for the directory. The catalog formatting is internal to Applewriter. Locking and unlocking are done by reading, then changing the attributes of a file. Renaming, deleting, setting prefixes, finding volumes on line, or creating a subdirectory are done directly with their respective ProDOS commands. Initiating a new disk is done by loading a separate formatting program, then jumping to that program. The formatter destructively overwrites the glossary, WPL, and any footnotes.

Finally, the printer commands are a set of internal routines that let you set the baud rate, word length, stop bits, and parity on a IIc. This routine also defeats video echo and suppresses any carriage returns that may be generated by the interface hardware.

[P] updates the print/program file or carries out a WPL command. A valid two-character, print/program value is converted to hex and entered in the correct slot in the print/program file. Absolute values are entered as such. Relative values are added to or subtracted from the old value. Two's complementing is used for subtraction. On TL and BL entries, the string is placed in the correct file. On UT, the underline token is saved. On NP, CP, and WPL commands, the selected command is completed.

[Q] accesses the additional functions menu. Binary tab and print/program values are loaded or saved as called for, using the ProDOS MLI. All of these values go in their respective stashes in main memory. Glossary or WPL text loads and saves are done similarly. The carriage return toggle sets or clears a display flag. The status toggle is identical to [esc] and may be replaced with something useful. Connecting printer to modem gives you a limited way to type directly to your printer. More importantly and more usefully, this selection also lets you send or receive text files over a modem.

A submenu on the [Q]-I selection lets you activate these modem features, such as recording incoming modem data or filtering control commands. The Quit option is surprisingly sophisticated and provides for an orderly exit to some other ProDOS system application program. Quitting includes reconnecting all disconnected video cards, and closing out current ProDOS activity in an orderly way.

[R] toggles the replace mode flag \$F5. When in the replace mode, a character is deleted from HIFILE before each character entry, then the new character is entered into the top of LOFILE as usual. The combination of deleting the cursed character and entering another character at the



cursed position gives the illusion of replacing the old character. Replace mode is aborted on practically all cursor motions.

[S] is the save command. On any save, the entire text is first moved to LOFILE. Then all or delimited portions of the text are moved to a sector buffer in main RAM at \$B700. Full sectors are transferred to disk as they are filled. Should appending be needed, ProDOS markers are set to allow adding to the end of an existing file rather than overwriting the previous bytes.

[T] sets or clears tabs. On a purge, the entire tab file is cleared to all zeros. On a Clear, only one pair of tab entries is set to zero. On a Set, the present position since the last carriage return is placed in the tab file. Up to 64 tabs are allowed. A separate tab status display is updated, causing all set tabs to appear in inverse and all cleared tabs to appear as normal. Although the status display only goes to 240 columns, tabs themselves can well exceed this number if the current paragraph is long enough.

[U] is the right arrow or frontspace. When it is the only key pressed, it moves the cursor forward one location by moving one character from HIFILE to LOFILE. With [closed apple], the right arrow does an express by-word frontspace, continually going forward until the first space is found. With [open apple], the right arrow retrieves a character from the swallow buffer instead of from HIFILE, placing the character in the top of LOFILE, and decrements the round and round swallow buffer pointer \$AC.

[V] toggles the verbatim flag \$72. With this flag set, all control characters except [M] or [V] are entered directly into the text file. This allows *imbedded* control characters for such things as special printing or typesetting commands. With the V flag cleared, control characters are used in their normal manner.

[W] inserts or deletes a whole word, depending on the data direction. On <, a word is saved to the word and paragraph deletion buffer starting at the first open spot available. Characters are removed from the top of LOFILE and placed into this buffer until either a space or an empty file is found. On >, a word is recovered from the word and paragraph deletion buffer, putting the characters in the top of LOFILE and stopping on a space. A round and round pointer pair \$94,95 keeps track of positions in the deletion buffer. A separate deletion overload counter makes sure the buffer does not overflow.

[X] is similar to [W] but [X] inserts or deletes an entire paragraph, keying on a carriage return rather than a space. On both [W] and [X], if [closed apple] is also used, the word or paragraph is saved to file but is not deleted from the text. This is most useful for copying short blocks of text.

[Y] is the screen splitting switch. On a [Y]-Y, the split screen is set up, using only 12 lines per display rather than the usual 24. One side of the split screen is active at a time. The other side is a static display of the way things were. Pointer \$F8 decides which side is active. On a [Y] with a split screen, control flips over to the other screen side by toggling \$F8. On a [Y]-N, the pointer is cleared, allowing the normal full screen display.



[Z] toggles the wraparound flag at \$E1. Wraparound is always present in the text file because each screen line ends with a high ASCII marker. If this flag is active, the screen update code ends each line on these markers. If wraparound is not necessary, characters are put on screen as they occur, stopping at 79 screen characters. The character slot to the extreme right is always reserved for a possible carriage return symbol, whether or not it is used. Note that full word breaks must be used if more than 80 columns are active.

[\_\_\_] calculates the page/position display. This routine is cumbersome and slow but also is most useful. Because operation is too slow for real time, you must toggle [\_\_\_] on only when you want specific page/position information. The routine works by counting carriage returns and comparing them to the printable lines per page. The total carriage returns are divided by the printable lines per page. The result gives you the page, and the remainder gives you the position on the final page. The need for numeric division causes the slowness.

We aren't quite through with control commands because I have saved two of the heavies for last. As a reminder, we are scanning through the various features of this program to see roughly what they do. Much more detail is found in Listing C.8 and in your own torn disassembly and cross reference.

Our first heavy is . . .

## Printing

Unlike some word processors, the ProDOS Applewriter 2.0 printing routines are part of the machine-resident editing code rather than a module separately loaded off the disk. In Applewriter, you have a choice of four possible print destinations. You can print to a real printer to get a hard copy. You can print to a modem or a special Ile plug-in card. You can print to the screen to see exactly what your printed text will look like, or you can print directly to a disk text file.

The last option gives you a document in final form, without any imbedded commands, that looks exactly like the document to be sent to the printer. Printing to pd8 is particularly useful when you are typesetting, need camera-ready copy, require multiple columns, want multiline headers or footers, or are transmitting between two different brands of computers. In fact, if anything seems like it cannot be done with Applewriter, chances are that a trip through pd8 land will bail you out one way or another. Once you decide what you want, WPL can make the whole thing invisible and automatic.

One gotcha: Be sure to have a unique filename for your pd8 images!

Otherwise pd8 files will get mixed up with your files that contain embedded commands and will royally foul the works. I often use a generic ZZZ for any temporary use of a pd8 file. The print destination is

specified with the `pd` command. A `pd0` outputs to the screen for what-you-see-is-what-you-get previews. A `pd1` dumps to a printer card in the selected slot. Rarely a `pd2` or `pd4` could be used to dump to a modem or some other special card. A `pd8` dumps directly to the disk.

Printing begins by moving everything to `LOFILE` with a `[E]` command. The printing pointer pair `$90,91` then moves up through the text file by starting at `$0801` and grabbing one character at a time.

Pages are formatted using the `print/program` values, such as top margin, left margin, right margin, bottom margin, page numbers, etc. At the beginning of the first page, the `pn` page number is saved to the running page counter pair at `$BE,BF`. The default left and right margins are saved as well. This way, the top and bottom line formats will stay the same throughout the document. The top line, if used, is formatted and printed first. This is done by reading the three possible delimited pieces out of the top line file and then moving them into a work area where the page number can be substituted for the `#` symbol.

Each left, center, or right piece is moved to a line buffer that has been previously filled to all spaces. The left piece starts at the left. The center piece starts half way across minus half the length of the center text. The right piece begins shy of the right margin by its length. After the top line, the top margin padding is put down, followed by the body of the page. The body is formatted and printed one line at a time, allowing for paragraph margins or outdents on the first line in each paragraph. Each line begins by getting enough characters out of the text file to fill the line.

As the characters come in, they are filtered for imbedded commands and for footnotes.

Imbedded commands start with a carriage return followed by a period followed by two or more letters. If these commands are found, the printing stops long enough to let the imbedded command do its thing. For instance, on an `.lm+5` command, printing halts momentarily. The left margin is retrieved, decimal five is added to it and then the left margin is replaced. The new left margin value will be picked up on the next line.

As you most likely have found out by now, any command that Applewriter does not recognize is treated as printable characters. This leads to the shortline problem. We have seen a `STRETCHIFIER` patch described in Chapter 6 that cures this hassle.

Characters are also filtered for footnotes, which begin with the `(<` command. If footnotes are found, they are stored in the footnote buffer at `$1400`, and the footnote flag `$FE` is set. This flag is incremented once for each footnote.

The very first footnote knocks two counts off the available number of printed lines. Any additional footnotes knock off one extra line. This gives a space between the bottom body line and the first footnote line.

At print time, any user separators `[__]` are automatically converted to `NULL` commands. That conversion works fine if you need `NULLs` for an old Epson. It is terrible if you need a user separator for a daisywheel HMI

command, a modem activity command, or for expanded printing on some newer dot matrix printers. A fix for this is described in Chapter 6.

At any rate, characters are gotten and filtered until enough whole words are entered to fit between the left and right margins. These characters are placed into the line formatting buffer at \$1C00. That line is then justified. Should left justification be in use, nothing more is done. All of the words remain flush left.

If center justification is in use, the length of the entered characters is subtracted from the line width. This new length is halved and then that number of spaces is used to offset the characters in the line buffer.

If right justification is in use, the length of the character string is subtracted from the line width, and that number offsets the characters in the line buffer.

In any of these three modes, you end up with the buffer holding the line justified in the correct position. Spaces are added as needed before the center justified and right justified text. Spaces are not needed *beyond* any text because the carriage return completes the entry. A row of printed spaces looks the same as the unprinted page, so trailing spaces are neither needed nor used.

On the fill justification of a long line, the needed number of padding spaces is calculated. Text is then moved one space to the right, beginning with the first space and repeating as often as needed to force the fill justification.

Microjustification is not available inside stock Applewriter.

Instead you use imbedded commands to tell an intelligent printer to microjustify for you. Naturally, if your printer has full microjustification available internally, your text will look much better than text justified by whole spaces. As we've seen, the enhanced Diablo 630 microjustifies beautifully.

Regardless of the justification mode, all of the characters end in the correct place in the line justification buffer. When the justified line is output for printing, it is preceded by enough spaces to make the left margin.

On first paragraph lines, the pm value is used to adjust the needed number of leading spaces.

As the line is printed, the characters are filtered for the underline token. Should this token appear, it is replaced with a space, and the underline mode flag \$E0 is toggled. Underlining is done by printing the underline character and then backing up one space and printing the character to be underlined. Underlining will not work on some very old or otherwise primitive dot matrix printers. The printer must be able to recognize the \$88 ASCII backspace command for this type of underlining.

As we have seen, underline is best left to the printer. This is done by imbedding suitable commands to turn the *printer's* underliner on and off when needed.

As many lines as are asked for are put in the body of the text. When finished, any footnotes are recovered from the footnote buffer and printed.



They are followed by the bottom line padding, and, if used, the bottom line.

Note that the stock program allows only a single top or bottom line. However, with repeated trips through pd8 land, you can have any number of top and bottom lines. You can also single space the headers and footers while double spacing your main text, as well as using even-odd headers. Good old pd8 will also let you do space-and-a-half and similar tricks.

Printing continues until all of LOFILE has been printed. At that point, a new file can be loaded and a cp continue printing command can be given, picking up exactly where you left off. The same running page number and current margin settings are kept. On the single sheet option, printing halts at the bottom of the page long enough for you to change paper.

By the way, if your IIe printer card does not defeat video echo, it will trash the screen and may slow things down, particularly at higher serial baud rates. The IIc serial interface automatically defeats any screen echo when you set the printer interface with [O]-J. As a reminder, special patches may be needed for intelligent IIe printing cards.

You will, of course, get the best printing with an intelligent printer or typesetter that accepts imbedded commands and can do its own proportional spacing, boldface, italics, shadow printing, and microjustification.

So much for printing. The real biggie is . . .

## WPL

WPL is a supervisory language that looks like a cross between PASCAL and assembler. Its intended use is as an executive controller that will handle long and involved tasks for you. Obvious uses are printing a multiple file book chapter with the correct headings and footings, customizing a mailing to a separate address list, counting words, putting down menus, prompting operators, building an index, etc.

But it's the non-obvious uses of WPL that boggle the mind.

The amazing thing about WPL is how much is done with how little. The additional code needed is rather short and compact. I have used WPL to insert or remove the line numbers from assembly code and to *picture process* strings sent to a plotter. I have used WPL to trick a printer into doing camera-ready copy and to handle automatic formatting. I have also used WPL to create high level graphic images. In fact, I am convinced that WPL is far more powerful at processing pictures than it is at words. Others have even written adventures in WPL.

WPL interfaces beautifully with Postscript, the typesetting language used on the Laserwriter.

We already saw how to use WPL to completely format a document for full bells and whistles superior quality printing. The message is over-

whelming. WPL is super powerful and super important. Without this language, Applewriter may have some second rate competition.

With WPL, that's all she wrote . . .

If you do not thoroughly know and aggressively use **WPL**, you are passing up at least 98 percent of the good stuff you can do with Applewriter.

So get with it. Now.

Figure 7.4 summarizes a WPL instruction. Each WPL instruction is one line long and ends with a carriage return. Lines are done normally in the order they are found in a WPL program although several important exceptions exist.

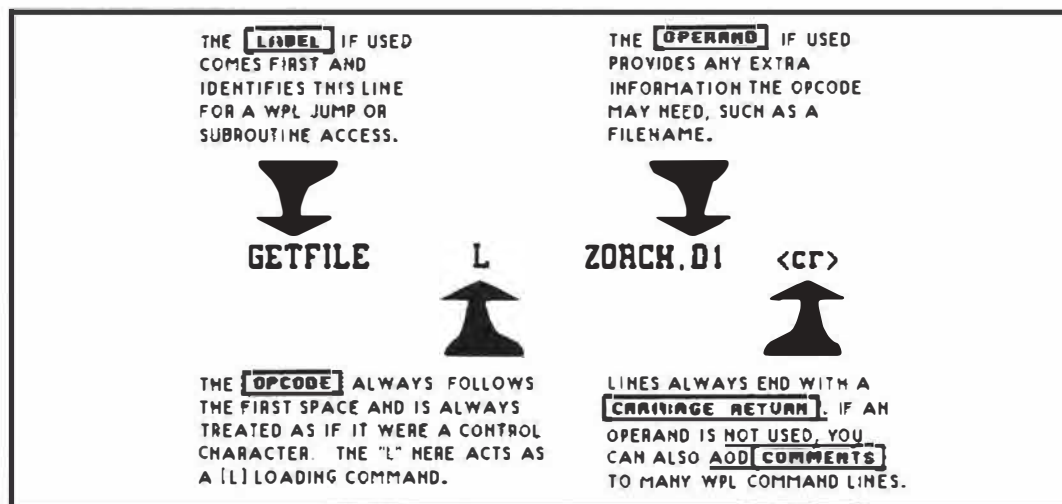


Fig. 7.4. A WPL command line is very similar to a line of assembly source code.

Each WPL line may begin with a label. The label must not have any spaces. If a label is used, it lets WPL find a certain line for possible jump or subroutine access.

If a label is not used, a space must be the first character on a WPL line. Either way, the first character *after* the first space in a WPL line is treated as

if that character were a control character. WPL then behaves just like you typed that control character from the keyboard. For instance, the WPL interpreter would see a line that consists of a space followed by a B as a [B] and would move the cursor to the beginning of the screen.

Although WPL lets you use lots of spaces for pretty printing, you can run out of program room fast if you try this. Thus, most non-trivial WPL programs are usually done in a compact and hard to read form.

Nearly anything you can do at the keyboard, WPL can do for you, automatically, potently, and without errors. Think of WPL as a high level language that is extremely good at editing long strings of characters and acting on them plus being a competent disk and printer supervisor.

So what is WPL and how does it work?

To answer, we first need a way to write a WPL program. Because a WPL program is nothing but some processed words, you write your WPL program on Applewriter, just like any old text file, and save it to disk.

One WPL command is called *do*. To *run* your WPL program ZORCH, you simply enter *[P] do ZORCH*. That is all there is to it. The *do* code first clears all the various WPL flags and work areas. This code then loads the named program into a WPL program file starting at \$1000. The program can be 1024 characters long if footnotes are in use or 2048 characters if not.

Note that you can beat the 1K or 2K character limit so that a WPL program can be arbitrarily long. Chain any number of WPL programs together end-to-end with *do* commands. You can also use one main WPL supervisory program to control several others. The others go back to the supervisor after carefully setting a variable or two to tell the supervisor where the program left off.

Variables are preserved when WPL programs are chained or otherwise linked together.

The *do* command also sets the WPL activity flag \$DF so that keystrokes will be read from the WPL file rather than from the keyboard. If the WPL flag is set, the first line of the WPL program is read. If a label is present, it is passed over, and WPL finds the first character beyond the first space or string of spaces. This character is converted into a control command and is processed just the way any control characters entered from the keyboard would be. Any remaining characters on the line are used as needed by the control command. For instance, a filename might follow an L for [L]oad, but a search and replace string might follow a F for [F]ind.

The WPL lines are read one at a time, usually in sequential order. Each line terminates with a carriage return. The final WPL line ends with a \$00 marker, which stops WPL and returns control to the keyboard.

WPL has jumps and subroutines. The WPL command *go* will start at the beginning of the WPL file and search for a label. On the jump command, that label is found and the program unconditionally jumps to that line and then continues from there. The WPL command *sr* does almost the same thing for subroutine access. The only difference is that a return address is remembered on a WPL stack at \$1D00, along with a stack



pointer \$92 that remembers where to return to. Returning is done when a RS command is found.

Subroutines can be nested to a depth of 32.

WPL has three numeric variables named (x), (y), and (z). Each can range from 0 to 65535. Any time an (x) is found, the value assigned to (x) will be substituted, and the same goes for (y) or (z). You can set these numerics to any value, either absolute or relative.

You can easily test a numeric for zero. With some hassle, you can also test a numeric for most any nonzero value. For instance, psx45 puts a decimal 45 into (x). psx7 sets (x) unconditionally to decimal seven. A command of psx+7 adds seven to whatever was already in (x). Most importantly, the command psx-1 decrements a counter loop involving (x) by a single count. The numerics are really nothing but print/program values and are stashed in the print/program file, such as lm or ut. See Listing C.3 for the exact locations.

Substitutions are done at the time the WPL line is interpreted.

WPL has string variables. Four of them are named \$A through \$D. These are stashed in the work files, starting at \$1E00. Just like the numerics, the strings are substituted for their symbols at the time the WPL is interpreted. Strings may be loaded from memory or disk with the ls command assigned to an immediate value with the as command and compared with the cs command. Check Listings C.3 and C.8 for more details.

During disk access, the pls load string command borrows an unused portion of the text file immediately *above* LOCURS out in no man's land. Because the \$A-D strings are allowed to be only 64 characters long, there is little danger of crashing into HICURS, except on a nearly full text file. String loader is done in the text file run to give all of the powerful loading options to WPL strings that the usual text loads receive. After use, the string above LOCURS is zeroed out so that this string does not become an unintentional part of the text file.

WPL has conditional execution. This is an absolutely essential feature of any computer language. The next WPL statement is skipped if a numeric reaches zero, if [F]ind cannot, if [L]oad will not, or if sc does not compare. The skipped statement is usually a jump, a subroutine call, or a program quit. Thus you can make a test and cause WPL to pick two different routes, depending on the result of that test.

WPL interacts with the user. You can clear the screen or print fixed screen messages with the ppr command. You can get a string from the user with a pin command. The display can be turned on with the pyd command and off with the pnd command. An off display computes much faster, besides holding the last prompt or message for you.

A pep command in WPL enables the printer if its value is not zero. You use this command to print only the page you want in the middle of a document. To accomplish this, put an .ep0 at the beginning of your document and an .ep1 where you want the actual printing to start.



Note that the command dot in your text file is the same as a [P] on the P in a WPL command. Thus [P]ep1 lets you turn the printer on. An .ep1 will turn the printer on when imbedded in the text. Finally, a pep1 from inside a WPL program will do exactly the same thing.

Many beginners do not pick up on the power of the STARTUP feature. On a cold boot, Applewriter looks for a WPL program named STARTUP. If Applewriter cannot find this file, things continue normally, without any error messages and without any retries.

If Applewriter STARTUP is present, anything in that file gets done in the intended order. Important things a startup program can do are set the prefix to drive two and give you a catalog of your work files. Startup programs can also load your glossaries, tab, or print files as needed. Fancier STARTUP programs can give you a help screen and menu selector that automates printing of multicopy, multisection documents, load and save boilerplate, control help screens, and do lots of other really neat things.

Here are a few . . .

#### WPL Tricks That Beginners Miss

---

Clear screen	ppr[L]
Beep	ppr[G]
Tweedle	ppr[G][G] [G]
Catalog to file	oa# pxxxxx
Set prefix	oh,d2
Turn printer off	.ep0
Turn printer on	.ep1

Clear screen works only after a .pnd command. Use the beep when an error occurs. Use the tweedle to get the user's attention when a long routine is finished.

WPL, of course, can have errors.

Lots of different things can go wrong with a WPL program. You might have a label missing or spelled wrong. You might be calling subroutines without returning. The program might get too long. Just as DOS has an orderly exit method and prompting for DOS errors, a separate WPL error processor shuts down WPL in an orderly manner and prompts the user. This error processor starts at \$40F6 in the AWD.SYS version of ProDOS Applewriter 2.0. [esc] shuts down a wayward WPL program. This same key also serves as a printer panic button.

WPL is great for what it is and what it does. But it is a specialized language and, as such, has some serious shortcomings. Its arithmetic capabilities are limited. A floating point multiply takes more than 46 seconds—that's seconds, not milliseconds or microseconds. Some other tasks can also be excruciatingly slow, particularly sort routines.

And that just about gets us one pass through Applewriter. Should you want all of this chapter in machine readable form, it is available as an eight disk ProDOS Applewriter Cookbook package, separate from the companion disk to this book. More details on the last few pages.



---

# 8

---

## **Capturing ProDOS Applewriter Version 2.0 Source Code**

Customizing ProDOS Applewriter 2.0,  
capturing source code,  
adding multiple columns with WPL,  
and a final wish list  
to wrap things up . . .

---



## Customizing ProDOS Applewriter 2.0

Obviously, you want to modify either all of ProDOS Applewriter 2.0 or part of WPL, or you wouldn't have gotten this far.

Okay.

First, you'll want to completely tear apart the program and thoroughly understand it.

Secondly, be sure that what you want done cannot be easily handled by a WPL routine of some kind that works on the stock code. Don't forget that amazing things can be handled with a WPL routine that autoboots on power up.

Listing C.9 shows two different ways to customize ProDOS Applewriter 2.0. In the personal method, you make any changes you like and permanently install them onto a third or higher backup copy of the original code. We saw lots of examples of how to use the personal method in Chapter 6. This personal method is simple and quick, but the result cannot be commercially sold or otherwise passed on. At least not legally. Also bad is the fact that a program has been changed without any renaming.

The preferred commercial method uses a booting .SYS program of your own. .SYS first calls for and loads the user's legal copy of the stock factory program. Then your .SYS program installs your patches and expansions in the machine resident code and runs the patched version. This method seems fully legal and has no duplicate filename problems. You are, of course, required to have a ProDOS license if you sell or pass on *any* code of *any* type that works under ProDOS. Apple's fees on this sort of thing start at \$50 annually.

Your custom booting code has three subtle gotchas: First, although the program can be in any language, the code must be a .SYS file and must be the very *first* .SYS file on the disk. Note that you can get between a .SYS file and any other by using the Type command, such as *BLOAD AWD.SYS, A\$2000, E\$6020, TSYS, D2*. In this example we grabbed a system file and installed it as a binary image.



Second, the MLI access part of your code that installs AWD.SYS must specify and set aside a ProDOS buffer at \$BB00 in main memory. Note that this is the only funny thing that the stock booting code does.

Third, be sure to verify that the correct version of the proper code is installed before you overwrite it. Very strange and wonderful things will happen if you install an Applewriter patch in the middle of Zork or Visicalc.

By the way, you can boot AW.SYSTEM from BASICS.SYSTEM by doing a PREFIX.D1, followed by a AW.SYSTEM.

Just about any attempt at modification should preserve the position and length of the stock code . . .

If you modify Applewriter, try to keep the position and the length of each code module *exactly* the same as it is in the original program.

If you do not match each original module, you will introduce all sorts of sticky complications. As you might gather from this rule, you should hold off trying to use the HIRES screens *in their intended locations* for anything unless you are ready to change things in a very big way.

The usual way to handle a module that gets longer is to put a jump in the module, then do your patch in the free part of main RAM from \$6020 to \$BF55. Lots of room is available.

When finished with your module, you either jump back to the stock code or RTS to it as a subroutine return. It is also best to provide compatibility with only one version per patch. At the present time, the best choice of a program is AWD.SYS, intended for the 80 column IIc or the expanded 128K IIe.

To repeat, leave everything exactly where it is in memory, making changes that overwrite, rather than relocate, code modules.

We saw a good way to add PEEK and POKE to older Applewriter IIe in *Enhancing Your Apple II*, Volume II (SAMS #22425). Unfortunately, ProDOS has no simple or easy way to add PEEK and POKE. Should you really need something BLOADED off disk, you need to set up a ProDOS MLI interface. Cloning or diverting the tab loading routine [Q]-A must be one feasible route.

## Why Modify?

Disclaimer time: All computer programmers face a dilemma.

If they do not provide ways to extend and change the language, their

language is forever theirs and its integrity is never compromised by things that others try to do to it.

If they do provide ways to extend or change the language, the limits can be pushed and fantastic and unexpected things can be done as many different people put their skills and thought processes to work improving and upgrading.

The same is true of a word processing program. If you change it, it may get better or it may get worse. If you do not change the program casually, it does exactly what is expected of it, reliably and certainly. If you do change the program, it may do great and wonderful things.

Then again, it might blow up in your face.

So, you are given both a powerful and a deadly ability when you attempt to modify a major program. Almost certainly, poorly thought out changes by a careless or inexperienced programmer will blow up the code outright, or else create subtle and infuriating bugs.

Needless to say, neither Sams nor I will clean up any of the mess you make. But have at it. Nothing ventured, nothing lost. Be sure to let us know what successes, if any, you may come up with.

## ***Thoughts on HIRES Dumps***

HIRES graphics dumps are reasonably easy to add to ProDOS Applewriter 2.0. The most serious problem is that a separate machine language printer driver is needed for virtually every make and model printer. Worse still, firmware routines in standard printer graphics dumper cards or modules cannot be used, unless such modules know how to do a "page three" dump. Even then, some customizing will be needed.

If you have studied Chapter 7 in detail, you found that the normal HIRES picture areas in main RAM from \$2000-5FFF hold all of the program itself. The similar area in auxiliary RAM sits smack in the middle of your text file. Thus, any attempt to put a HIRES picture in the machine where it belongs and actually could be viewed would be a real hassle. One solution is to use HIRES "page three", BLOADing your picture to be dumped in \$6000-7FFF in main RAM. A better solution is to use page "three-and-one-half," starting at \$7000-7FFF, which leaves room at the end of the stock AWD.SYS code for expansions and add-ons without any memory conflict.

Three ways are available to handle graphics images. The first is by way of *string precoding*. The second is with a *full screen dump*. The third is by *post processing*.

With the string precoding method, you use some other program, written in machine language or even Applesoft, to take a HIRES image and convert it to a text string of characters that will fire the dot matrix pins or move the daisywheel in the right direction at the right time. You get the best results on a dot matrix printer, and then when you fire only the middle

six pins. Your pre-converted HIRES image is loaded as a *text* file, exactly as any other text file would be.

The advantage of this method is that it can work on stock ProDOS Applewriter 2.0, even on the short AWC.SYS version. Disadvantages include the need for highly custom code for each and every printer and a fearsome gobbling up of available text file space. If you try this method, pick something simple like a logo or a letterhead and see what you can come up with.

With the full-screen dump method, you expand the print/program commands to include an *hd/pixname/* command. When this command is found, the word processor stops, loads a HIRES image into, say \$7000 8FFF and then dumps the image to the printer. Word processing resumes after the dump as if nothing happened. Similarly, *hd/paramname/* could set HIRES parameters, such as normal or inverse, standard or double resolution, portrait or landscape, 1X versus 2X, left margin, vertical space reserved, or whatever. Yet another imbedded command would let you load the dumper code needed for your particular printer.

Note that just about every printer will need its own custom code. Yes, you can do graphics on a daisywheel printer. In fact, if you use a "square period" as your printing element, you can end up with graphics quality that is much better than dot matrix.

Another myth bites the dust.

With post processing, you first do a pd8 and then use *some other* program to read and print the text file. Printing can stop at any time to allow a HIRES dump or anything else you want. Important advantages of post processing are that only limited programming skills are needed and that virtually anything can be done with an oddball printer. Post processing, however, can be very slow.

Similar techniques could also be used to do a wall-to-wall microjustify on the Imagewriter or some daisywheels. Instead of loading a HIRES picture, you download a table of proportional space values. Note that this is not needed for the full microjustification we did earlier on the enhanced Diablo 630.

The biggie in this module involves . . .

## ***Capturing Your Own Source Code***

Needless to say, it is far easier to make heavy changes in any program by working directly with source code than by puzzling over mysterious object code. Captured source code is almost essential for such heavy reworking as repositioning the program or extending modules. Source code also may be necessary if you are to integrate your word processor with other program modules, such as a spreadsheet, data base manager, or telecommunications package.

A good many reasons exist to *not* try and capture source code.



For one thing, a lot of time, patience, and effort are involved if you are to do the job correctly. More importantly, things get sticky fast if you try to make any commercial use of your captured and modified source code. You are, of course, free to make any changes you want any way you want to any program you personally own, as long as you do so for your own use only. The problems begin if you try to sell or otherwise pass on "your" work, which is really a mix of the value added by what you have done and the value produced by the original author.

Play fair or don't play at all.

I'll show you a fully automatic source code capturer for version 2.0 of AWD.SYS. All you do is push the button and sit back and watch. Because this capturer is fully automatic, though, any change whatsoever in AWD.SYS will foul up the works, and you will end up back on square one.

You will need an intelligent disassembler program for source code capture. The one I use is . . .

### **An Intelligent Disassembler**

---

DISASM 2.2e from  
RAK-WARE  
41 Ralph Road  
West Orange NJ 07052  
(201) 325-1885

Note that this particular version of this specific disassembler must be used for these instructions to work. The capture process works best under DOS 3.3e rather than ProDOS. Done this way, less sand gets kicked in your face, and you will get better results faster.

Because the minimally labeled and totally undocumented source code is more than 10,000 lines long, it will not all fit on one disk. We will use a pair of disks to capture the source code for you, creating four sequential source code text files. Listing C.10 shows the capturing process. In addition to DISASM IIe and converted AWD.SYS, four special programs are needed. These programs are named HELLO, GRABBER.D.12, GRABBER.D.34, and SNEAKY.D.

HELLO is shown in Listing C.11. This program is a stock Applesoft program that prompts you to create a pair of disks. It then picks which grabber to execute.

GRABBER.D.12 is a text file that is run as an EXEC by HELLO. This dude sits on the first disk and grabs the first two parts of the source code, naming them AWD.SOURCE1 and AWD.SOURCE2. GRABBER.D.12 appears in Listing C.12.

GRABBER.D34 is a similar text file that sits on the second disk and grabs the remaining two parts of the source code, which are then named AWD.SOURCE3 and AWD.SOURCE4. Listing C.13 shows details.

Like every good magic trick, source code capture has to have a gimmick somewhere. Called SNEAKY.D, this listing is disgustingly elegant and makes the whole capture process possible without fatal errors. See Listing C.14 for info.

Naturally, I will save details on this one as an exercise for the student.

Neat, huh?

Anyway, once you create the needed disks, you simply press a key or two and sit back and watch a very long and very confusing screen display. When completed, out pops the source code, ready for your further rework. The relation between the four captured source code modules are shown in Figure 8.1.

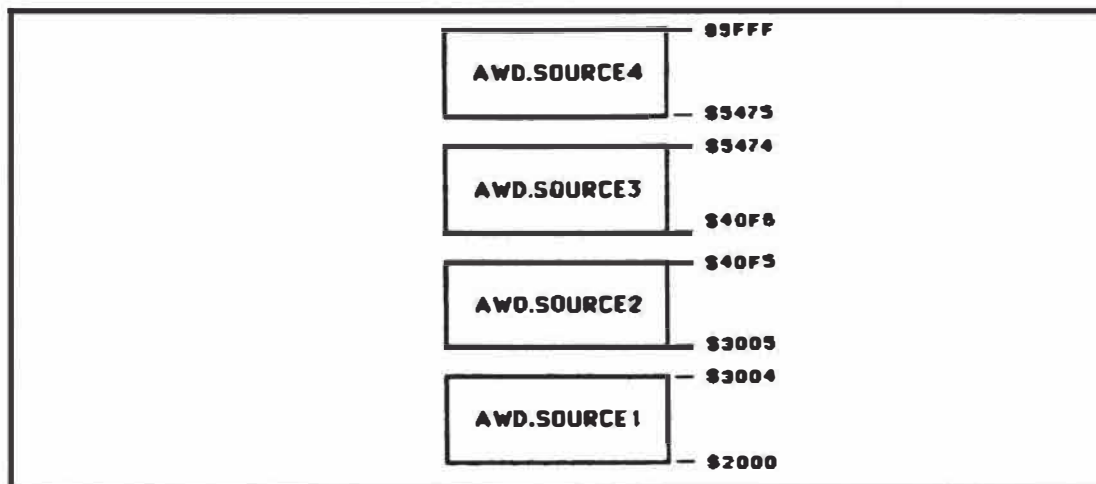


Fig. 8.1. The captured AWD.SYS source code ends up in four pieces.

A most important capture rule . . .

Captured source code is totally and utterly useless if it does not *exactly* reassemble into code that is a *perfect* match to the original object code.

So be sure to reassemble your captured source code back into object code and compare it byte for byte against the original. Full details on working with source and object code appear in my *Assembly Cookbook for the Apple II and IIe* (SAMS #22331).

The usual way to verify assembled source code is with the monitor or verify [V] command. For instance, to verify AWD.SOURCE1, *BLOAD*

*AWD.SOURCE1* at \$2000, *BLOAD AWD.SYS* at \$4000, then do a *1FFF<3FFF5005V*. You should get two errors, one beyond each end of the code being verified.

## A Final Plum

A routine called *WPL.TWO COLUMNS* appears as Listing C.15 and can be used to format your printed hard copy into two or more vertical columns, as might be needed for a newsletter or a technical article. Yes, you can individually justify each column or even microjustify it. The code is intended as a core module that does the tedious and messy stuff only on a single page at a time. You can include this module in fancier routines of your own.

What you do is create your left column and your right column ahead of time, using *pd8* to precisely format each column in the exact way you want that column sent to your printer. Be sure any and all imbedded commands are removed.

Important: Use a zero left margin for both columns.

When you run this module, it sets a single tab at the intended position of your right column. Note that the distance between columns is measured from the left edge of the left column to the left edge of the right column. A separation value of 39 is sometimes a good starting point. After the tab is set, the two columns are loaded and the end of each line of the left column has a special marker appended. The beginning of the entire right column file has a separate special marker added.

The code then grabs the first line of the second column using [X]. The module then goes back to the beginning and finds the marker at the end of the first line of the first column. That marker is erased and a tab is inserted to move the text to the starting position of the second column. The first line of the second column is then returned from the [X] stash and is inserted into its proper place.

This puts two text strings onto one line, with enough spaces between the strings to align the right column. The process continues for each line.

Formatting takes around one second per line or somewhat under one minute per page. When you are finished with *WPL.TWO COLUMNS*, treat the result as a solid block of text and add your own headers, footers, and left margin. Two sample columns, which are named *SAMPLE.LEFT.COLUMN* and *SAMPLE.RIGHT.COLUMN* appear on the companion disk to this volume. Use these samples to get started.

For Applewriter's internal by-spaces justification, simply use *fj* on your original columns before you do your *pd8* print to disk. A full microjustification of both columns is hairy and very much depends on your printer, but text can be fully microjustified if your printer is bright enough.



For three or more columns, repeat WPL.TWO COLUMNS as often as you have to, treating everything done on the previous pass as your left column. The only limits are the width of your printer and Applewriter's 255 character line length.

## A Wish List

I personally feel that this latest version of Applewriter is the greatest ever.

Generally though, the better things are, the more that can be done to improve them. As a final wrap up, here is a wish list from all the people on all the helpline calls so far. The items on this list are things that can be done right now with lots of custom or detail work, but they would best be included in Version 3.0 or whatever.

We will continue to chip away at the list. We will also continue to support newer versions of this great word processor. Let us know what you need or want to see.

Here's the list . . .

1. Optional use of prefixes, much less memory hogging, long free form filenames, and some decent speed.
2. An init feature that does not trash the glossary or WPL.
3. Built-in STRETCHIFIER and user-definable NULL substitution.
4. Larger WPL and glossary storage areas.
5. Much bigger text file area.
6. More WPL variables.
7. A reasonably fast WPL sorting ability.
8. Built in multicolumn text abilities.
9. Soft hyphens and hard spaces.
10. Multicolumn catalog display options.
11. HIREs dumpers for most major printers.
12. Better Appleworks compatibility.
13. Microjustification for more printers.
14. Real time page/position display.
15. Internal speech links for the handicapped.
16. More and improved cursor motions.
17. Capability of rerunning WPL without going to disk.
18. Load to screen under WPL.
19. Full compatibility with most IIe printer cards.
20. Improved modem abilities, particularly dialup.
21. Co-resident EDASM and/or limited Applesloth access.
22. Simple PEEK and POKE ability.
23. Improved WPL scanning from the cursed character.
24. Improved *not*, *and*, and *or* WPL logic.
25. Logo, letterhead, and icon graphics ability.

26. Erase-to-end and erase-to-beginning on one keystroke.
27. Multiline headers built in.
28. Space and a half.
29. More powerful footnotes and footnoting at end of document.
30. Author's keyword indexing.
31. Even-odd pagination.
32. Better compatibility with third-party video cards.
33. A limited capability of running on the II+ and clones.
34. Laser printer interface, including POSTSCRIPT.
35. Internal clock-calender interface.
36. Formal link for user-defined modules.

That's all, folks . . .



---

# A

---

## **WPL Programs and Applesoft Patches**

**Program A.1.** *Running this AWIle NULLIFIER Applesoft program will automatically modify your AWIle diskettes, so you can imbed NULL characters into your text files.*

```

100 REM
110 REM *****
120 REM *
130 REM * "NULLIFIER" FOR *
140 REM *
150 REM * APPLEWRITER IIe *
160 REM *
170 REM * VERSION 1.0 *
180 REM *.....*
190 REM *
200 REM * COPYRIGHT 1984 BY *
210 REM * DON LANCASTER AND *
220 REM * SYNERGETICS, BOX *
230 REM * 809 THATCHER AZ. *
240 REM * 85552. 602-428-4073 *
250 REM *
260 REM * ALL COMMERCIAL *
270 REM * RIGHTS RESERVED *
280 REM *
290 REM *****

300 REM This program modifies a
310 REM backup copy of AWIle
320 REM so that NULL commands
330 REM can be imbedded as [@]
340 REM control characters.

350 REM Two uses of NULLs are
360 REM to provide superscript
370 REM and underline for an
380 REM Epson printer.

390 REM .....

400 TEXT : HOME : CLEAR
410 HIMEM: 8000
420 VTAB 1: HTAB 8:
    A$ = "Applewriter IIe NULLifier":
    GOSUB 830
430 PRINT : GOSUB 880
440 PRINT
450 FOR N = 1 TO 39: PRINT CHR$ (127);:
    GOSUB 870: NEXT N
460 GOSUB 880
470 VTAB 5: HTAB 1:
    A$ = "This program will patch Applewriter IIe":
    GOSUB 830: PRINT
480 VTAB 6: HTAB 1:
    A$ = "so that NULL characters can be imbedded":
    GOSUB 830:
490 VTAB 7: HTAB 1:
    A$ = "by using a [V][2][V] command.":
    GOSUB 830
500 GOSUB 880
510 VTAB 10: HTAB 4:
    A$ = "Patch ONLY your THIRD BACKUP copy!":
    GOSUB 830
520 GOSUB 880: GOSUB 880

```

## Program A.1—cont.

```

530 VTAB 14: HTAB 4:
    A$ = "Please put your THIRD BACKUP copy":
    GOSUB 830
540 VTAB 15: HTAB 4:
    A$ = "of AWIIE into Drive #1. Then push": GOSUB 830
550 GOSUB 880
560 VTAB 17: HTAB 12:
    A$ = "<SPACE> to CONTINUE": GOSUB 830
570 VTAB 19: HTAB 19: A$ = "-or-": GOSUB 830
580 VTAB 21: HTAB 13:
    A$ = "<ESCAPE> to ABORT": GOSUB 830
590 VTAB 23: HTAB 19: PRINT "< >-"
600 VTAB 23: HTAB 21: GET Z$
610 IF Z$ < > " " THEN 820
620 REM

```

## Check Validity

```

630 PRINT
640 PRINT "[D]BLOAD OBJ.APWRT][E,A$2300
650 IF PEEK (9956) < > 15 THEN 800
660 IF PEEK (18621) < > 214 THEN 800
670 PRINT "[D]BLOAD OBJ.APWRT][F,A$2300
680 IF PEEK (10116) < > 21 THEN 800
690 IF PEEK (18998) < > 214 THEN 800
700 POKE 10116,0: POKE 18998,0
710 PRINT "[D]UNLOCK OBJ.APWRT][F"
720 PRINT "[D]BSAVE OBJ.APWRT][F,A$2300,L$30D3"
730 PRINT "[D]LOCK OBJ.APWRT][F"
740 PRINT "[D]BLOAD OBJ.APWRT][E,A$2300"
750 POKE 9956,00: POKE 18621,00
760 PRINT "[D]UNLOCK OBJ.APWRT][E"
770 PRINT "[D]BSAVE OBJ.APWRT][E,A$2300,L$2F5A"
780 PRINT "[D]LOCK OBJ.APWRT][E"
790 TEXT : HOME : A$ = "IT WORKED!": GOSUB 830:
    PRINT : PRINT : PRINT : PRINT : PRINT : END
800 TEXT : HOME :
    A$ = "Will not verify as AWIIE; patch ABORTED":
    GOSUB 830: PRINT : PRINT : PRINT :
    PRINT : PRINT : END
810 REM
820 REM
830 REM

```

## Noisy screen machine

```

840 FOR N = 1 TO LEN (A$): PRINT MID$ (A$,N,1):
850 GOSUB 870: REM Clickety clack
860 NEXT N: RETURN
870 ZZ = PEEK (49200) + PEEK (49200):
    FOR M = 1 TO 17: NEXT M: RETURN
880 FOR N = 0 TO 700: NEXT N: RETURN

```

.....

Gotchas: Do NOT use this program on either "factory" diskette!  
Use only on your THIRD or higher backup copies!

In the above listing, [D] stands for "control-D"; all other brackets are real.

This program is available ready-to-run on the companion diskette.



Program A.2. *This AWIle STRETCHIFIER Applesoft program modifies your Applewriter IIe backup diskettes to eliminate the shortlines that result from imbedded printing commands involving escape sequences.*

```

100 REM

110 REM *****
120 REM *
130 REM * "STRETCHIFIER" FOR *
140 REM *
150 REM * APPLEWRITER IIe *
160 REM *
170 REM * VERSION 1.0 *
180 REM *.....*
190 REM *
200 REM * COPYRIGHT 1984 BY *
210 REM * DON LANCASTER AND *
220 REM * SYNERGETICS, BOX *
230 REM * 809 THATCHER AZ. *
240 REM * 85552. 602-428-4073 *
250 REM *
260 REM * ALL COMMERCIAL *
270 REM * RIGHTS RESERVED *
280 REM *
290 REM *****

300 REM This mod changes a
310 REM backup copy of AWIle
320 REM so imbedded escape
330 REM commands pass through
340 REM the justify routines.

350 REM This eliminates the
360 REM "shortline" problem
370 REM and lets you fully
380 REM use a fancy printer.

390 REM
    .....

400 TEXT : HOME : CLEAR
410 HIMEM: 8000
420 VTAB 1: HTAB 7:
    A$ = "Applewriter IIe STRETCHifier":
    GOSUB 980
430 PRINT : GOSUB 1030
440 PRINT
450 FOR N = 1 TO 39: PRINT CHR$ (127);:
    GOSUB 1020: NEXT N
460 GOSUB 1030
470 VTAB 5: HTAB 1:
    A$ = "This program will patch Applewriter IIe":
    GOSUB 980: PRINT
480 VTAB 6: HTAB 1:
    A$ = "to eliminate the short lines created by":
    GOSUB 980
490 VTAB 7: HTAB 1:
    A$ = "imbedded printer escape sequences.":
    GOSUB 980
500 : GOSUB 1030
510 VTAB 10: HTAB 4:
    A$ = "Patch ONLY your THIRD BACKUP copy!":
    GOSUB 980
520 GOSUB 1030: GOSUB 1030

```

## Program A.2—cont.

```

530 VTAB 14: HTAB 4:
    A$ = "Please put your THIRD BACKUP copy":
    GOSUB 980
540 VTAB 15: HTAB 4:
    A$ = "of AWIie into Drive #1. Then push":
    GOSUB 980
550 GOSUB 1030
560 VTAB 17: HTAB 12:
    A$ = "<SPACE> to CONTINUE": GOSUB 980
570 VTAB 19: HTAB 19: A$ = "-or-": GOSUB 980
580 VTAB 21: HTAB 13: A$ = "<ESCAPE> to ABORT":
    GOSUB 980
590 VTAB 23: HTAB 19: PRINT "-< >-"
600 VTAB 23: HTAB 21: GET Z$
610 IF Z$ < > " " THEN 970
620 REM
    Check Validity

630 PRINT
640 PRINT "[D]BLOAD OBJ.APWRT][E,A$2300
650 IF PEEK (14720) < > 235 THEN 950
660 IF PEEK (17396) < > 153 THEN 950
670 IF PEEK (17436) < > 252 THEN 950
680 PRINT "[D]BLOAD OBJ.APWRT][F,A$2300
690 IF PEEK (15063) < > 100 THEN 950
700 IF PEEK (17771) < > 153 THEN 950
710 IF PEEK (17811) < > 117 THEN 950
720 POKE 15062,96: POKE 15063,153
730 POKE 15064,00: POKE 15065,22: POKE 15066,201:
    POKE 15067,155: POKE 15068,208: POKE 15069,04:
    POKE 15070,230: POKE 15071,211
740 POKE 15072,230: POKE 15073,211: POKE 15074,96:
    POKE 15075,196: POKE 15076,220: POKE 15077,240:
    POKE 15078,14: POKE 15079,185
750 POKE 15080,00: POKE 15081,22: POKE 15082,201:
    POKE 15083,155: POKE 15084,208: POKE 15085,04:
    POKE 15086,198: POKE 15087,211
760 POKE 15088,198: POKE 15089,211: POKE 15090,136:
    POKE 15091,208: POKE 15092,238: POKE 15093,76:
    POKE 15094,117: POKE 15095,71
770 POKE 17771,32: POKE 17772,215: POKE 17773,58
780 POKE 17810,76: POKE 17811,227: POKE 17812,58
785 IF PEEK (20365) = 176 THEN POKE 20365, 182:
    REM RECONNECT HELP SCREENS
790 PRINT "[D]UNLOCK OBJ.APWRT][F"
800 PRINT "[D]BSAVE OBJ.APWRT][F,A$2300,L$30D3"
810 PRINT "[D]LOCK OBJ.APWRT][F"
820 PRINT "[D]BLOAD OBJ.APWRT][E,A$2300"
830 POKE 14719,96
840 POKE 14720,153: POKE 14721,00: POKE 14722,22:
    POKE 14723,201: POKE 14724,155: POKE 14725,208:
    POKE 14726,04: POKE 14727,230
850 POKE 14728,211: POKE 14729,230: POKE 14730,211:
    POKE 14731,96: POKE 14732,196: POKE 14733,220:
    POKE 14734,240: POKE 14735,14
860 POKE 14736,185: POKE 14737,00: POKE 14738,22:
    POKE 14739,201: POKE 14740,155: POKE 14741,208:
    POKE 14742,04: POKE 14743,198
870 POKE 14744,211: POKE 14745,198: POKE 14746,211:
    POKE 14747,136: POKE 14748,208: POKE 14749,238:
    POKE 14750,76: POKE 14751,252
880 POKE 14752,69
885 IF PEEK (19988) = 176 THEN POKE 19988, 182:
    REM RECONNECT HELP SCREENS
890 POKE 17396,32: POKE 17397,128: POKE 17398,57

```

## Program A.2—cont.

```

900 POKE 17435,76: POKE 17436,140: POKE 17437,57
910 PRINT "[D]UNLOCK OBJ.APWRT][E"
920 PRINT "[D]BSAVE OBJ.APWRT][E,A$2300,L$2F5A"
930 PRINT "[D]LOCK OBJ.APWRT][E"
940 TEXT : HOME :A$ = "IT WORKED!": GOSUB 980:
    PRINT : PRINT : PRINT : PRINT : END
950 TEXT : HOME :
    A$ = "Will not verify as AWIle; patch ABORTED":
    GOSUB 980: PRINT : PRINT : PRINT :
    PRINT : PRINT : END
960 GOTO 960
970 TEXT : HOME : CLEAR : END
980 REM
    Noisy screen machine

990 FOR N = 1 TO LEN (A$): PRINT MID$ (A$,N,1);
1000 GOSUB 1020: REM Clickety clack
1010 NEXT N: RETURN
1020 ZZ = PEEK (49200) + PEEK (49200):
    FOR M = 1 TO 17: NEXT M: RETURN
1030 FOR N = 0 TO 700: NEXT N: RETURN

```

.....

Gotchas: Do NOT use this program on either "factory" diskette!  
Use only on your THIRD or higher backup copies!

This patch gives an exact repair for two-character escape sequences. For three-character imbedded sequences, "bank" characters by using [esc][esc] to bank two characters, or [esc][esc][@] to bank one character.

In the above listing, [D] stands for "control-D". All other brackets are real.

This program is available ready-to-run on the companion diskette.

**Program A.3.** *This AWIIe PATCHIFIER Applesoft program modifies your Applewriter IIe backup diskettes to let you install custom run time modifications by using the [O]-C command.*

```

100 REM
110 REM *****
120 REM *
130 REM * "PATCHIFIER" FOR *
140 REM *
150 REM * APPLEWRITER IIe *
160 REM *
170 REM * VERSION 1.0 *
180 REM *.....*
190 REM *
200 REM * COPYRIGHT 1984 BY *
210 REM * DON LANCASTER AND *
220 REM * SYNERGETICS, BOX *
230 REM * 809 THATCHER AZ. *
240 REM * 85552. 602-428-4073 *
250 REM *
260 REM * ALL COMMERCIAL *
270 REM * RIGHTS RESERVED *
280 REM *
290 REM *****

300 REM This mod changes a
310 REM backup copy of AWIIe
320 REM so run-time patches
330 REM can be made with the
340 REM [O]-C command.

350 REM BEWARE! Careless use
360 REM of patches can destroy
370 REM both the program and
380 REM your textfiles.
390 REM
.....

400 TEXT : HOME : CLEAR
410 HIMEM: 8000
420 VTAB 1: HTAB 8:
    A$ = "Applewriter IIe PATCHifier":
    GOSUB 870
430 PRINT : GOSUB 920
440 PRINT
450 FOR N = 1 TO 39: PRINT CHR$ (127);:
    GOSUB 910: NEXT N
460 GOSUB 920
470 VTAB 5: HTAB 1:
    A$ = "This program will patch Applewriter IIe":
    GOSUB 870: PRINT
480 VTAB 6: HTAB 1:
    A$ = "so run-time custom modifications may be":
    GOSUB 870
490 VTAB 7: HTAB 1:
    A$ = "made using the [O]-C command.":
    GOSUB 870
500 GOSUB 920
510 VTAB 10: HTAB 4:
    A$ = "Patch ONLY your THIRD BACKUP copy!":
    GOSUB 870
520 GOSUB 920: GOSUB 920
530 VTAB 14: HTAB 4:
    A$ = "Please put your THIRD BACKUP copy":
    GOSUB 870
540 VTAB 15: HTAB 4:
    A$ = "of AWIIe into Drive #1. Then push":
    GOSUB 870

```



## Program A.3--cont.

```

550 GOSUB 920
560 VTAB 17: HTAB 12:
    A$ = "<SPACE> to CONTINUE": GOSUB 870
570 VTAB 19: HTAB 19:A$ = "-or-": GOSUB 870
580 VTAB 21: HTAB 13:
    A$ = "<ESCAPE> to ABORT": GOSUB 870
590 VTAB 23: HTAB 19: PRINT "-< >-"
600 VTAB 23: HTAB 21: GET Z$
610 IF Z$ < > " " THEN 860
620 REM [J]CHECK VALIDITY[J]
630 PRINT
640 PRINT "[D]BLOAD OBJ.APWRT][E,A$2300
650 IF PEEK (19840) < > 214 THEN 840
660 IF PEEK (19841) < > 229 THEN 840
670 IF PEEK (19842) < > 242 THEN 840
680 PRINT "[D]BLOAD OBJ.APWRT][F,A$2300
690 IF PEEK (20217) < > 214 THEN 840
700 IF PEEK (20218) < > 229 THEN 840
710 IF PEEK (20219) < > 242 THEN 840
720 POKE 20217,194: POKE 20218,236: POKE 20219,239:
    POKE 20220,225: POKE 20221,228: POKE 20222,160:
    POKE 20223,208
730 POKE 20224,225: POKE 20225,244: POKE 20226,227:
    POKE 20227,232
740 PRINT "[D]UNLOCK OBJ.APWRT][F"
750 PRINT "[D]BSAVE OBJ.APWRT][F,A$2300,L$30D3"
760 PRINT "[D]LOCK OBJ.APWRT][F"
770 PRINT "[D]BLOAD OBJ.APWRT][E,A$2300"
780 POKE 19840,194: POKE 19841,236: POKE 19842,239:
    POKE 19843,225: POKE 19844,228: POKE 19845,160:
    POKE 19846,208: POKE 19847,225
790 POKE 19848,244: POKE 19849,227: POKE 19850,232
800 PRINT "[D]UNLOCK OBJ.APWRT][E"
810 PRINT "[D]BSAVE OBJ.APWRT][E,A$2300,L$2F5A"
820 PRINT "[D]LOCK OBJ.APWRT][E"
830 TEXT : HOME :A$ = "IT WORKED!":
    GOSUB 870: PRINT : PRINT :
    PRINT : PRINT : PRINT : END
840 TEXT : HOME :
    A$ = "Will not verify as AWIie; patch ABORTED":
    GOSUB 870: PRINT : PRINT : PRINT :
    PRINT : PRINT : END
850 GOTO 850
860 TEXT : HOME : CLEAR : END
870 REM [J] Noisy screen machine
880 FOR N = 1 TO LEN (A$): PRINT MID$ (A$,N,1);
890 GOSUB 910: REM Clickety clack
900 NEXT N: RETURN
910 ZZ = PEEK (49200) + PEEK (49200):
    FOR M = 1 TO 17: NEXT M: RETURN
920 FOR N = 0 TO 700: NEXT N: RETURN

```

.....  
Gotchas: Do NOT use this program on either "factory" diskette!  
Use only on your THIRD or higher backup copies!

Incorrectly used patches can damage both the program  
and your work files beyond recovery, so BE CAREFUL!

"E" patches must be used only on OBJ.APWRT][E code.  
"F" patches must be used only on OBJ.APWRT][F code.

In the above listing, [D] stands for "control-D". All  
other brackets are real.

This program is available ready-to-run on the companion  
diskette.

**Program A.4.** *This AWIIe CLARIFIER Applesoft program modifies your Applewriter IIe backup diskettes to eliminate trashing of the IIc status display line.*

```

100 REM
110 REM *****
120 REM *
130 REM * "CLARIFIER FOR" *
140 REM *
150 REM * APPLEWRITER IIe *
160 REM *
170 REM * VERSION 1.0 *
180 REM * ..... *
190 REM *
200 REM * COPYRIGHT 1984 BY *
210 REM * DON LANCASTER AND *
220 REM * SYNERGETICS, BOX *
230 REM * 809 THATCHER AZ. *
240 REM * 85552. 602-428-4073 *
250 REM *
260 REM * ALL COMMERCIAL *
270 REM * RIGHTS RESERVED *
280 REM *
290 REM *****

300 REM This mod changes a
310 REM backup copy of AWIIe
320 REM to eliminate trashing
330 REM of the IIc status line.

340 REM This lets you use an
350 REM older version of AWIIe
360 REM on either a IIc or IIe.
370 REM
.....
380 TEXT : HOME : CLEAR
390 HIMEM: 8000
400 VTAB 1: HTAB 8:
    A$ = "Applewriter IIe CLARifier":
    GOSUB 910
410 PRINT : GOSUB 960
420 PRINT
430 FOR N = 1 TO 39: PRINT CHR$ (127);:
    GOSUB 950: NEXT N
440 GOSUB 960
450 VTAB 5: HTAB 1:
    A$ = "This program will patch Applewriter IIe":
    GOSUB 910: PRINT
460 VTAB 6: HTAB 1:
    A$ = "to eliminate trashing of the IIc status":
    GOSUB 910
470 VTAB 7: HTAB 1: A$ = "line.": GOSUB 910
480 : GOSUB 960
490 VTAB 10: HTAB 4:
    A$ = "Patch ONLY your THIRD BACKUP copy!":
    GOSUB 910
500 GOSUB 960: GOSUB 960
510 VTAB 14: HTAB 4:
    A$ = "Please put your THIRD BACKUP copy":
    GOSUB 910
520 VTAB 15: HTAB 4:
    A$ = "of AWIIe into Drive #1. Then push":
    GOSUB 910
530 GOSUB 960

```



## Program A.4—cont.

```

540 VTAB 17: HTAB 12:
    A$ = "<SPACE> to CONTINUE": GOSUB 910
550 VTAB 19: HTAB 19: A$ = "-or-": GOSUB 910
560 VTAB 21: HTAB 13:
    A$ = "<ESCAPE> to ABORT": GOSUB 910
570 VTAB 23: HTAB 19: PRINT "-< >-"
580 VTAB 23: HTAB 21: GET Z$
590 IF Z$ < > " " THEN 900
600 REM
    Check Validity

610 PRINT
620 PRINT "[D]BLOAD OBJ.APWRT][E,A$2300
630 IF PEEK (14472) < > 188 THEN 880
640 IF PEEK (14709) < > 41 THEN 880
650 IF PEEK (14753) < > 57 THEN 880
660 PRINT "[D]BLOAD OBJ.APWRT][F,A$2300
670 IF PEEK (14815) < > 188 THEN 880
680 IF PEEK (15052) < > 41 THEN 880
690 IF PEEK (15096) < > 59 THEN 880
700 POKE 14815,60: POKE 14816,36: POKE 14817,207:
    POKE 14818,16: POKE 14819,2: POKE 14820,169:
    POKE 14821,62
710 POKE 15052,208: POKE 15053,42
720 POKE 15062,96
725 IF PEEK (20365) = 176 THEN POKE 20365,182:
    REM RECONNECT HELP SCREENS
730 POKE 15096,41: POKE 15097,127: POKE 15098,201:
    POKE 15099,96: POKE 15100,176: POKE 15101,208:
    POKE 15102,201: POKE 15103,64
740 POKE 15104,144: POKE 15105,204: POKE 15106,41:
    POKE 15107,63: POKE 15108,176: POKE 15109,200
750 PRINT "[D]UNLOCK OBJ.APWRT][F"
760 PRINT "[D]BSAVE OBJ.APWRT][F,A$2300,L$30D3"
770 PRINT "[D]LOCK OBJ.APWRT][F"
780 PRINT "[D]BLOAD OBJ.APWRT][E,A$2300"
790 POKE 14472,60: POKE 14473,36: POKE 14474,207:
    POKE 14475,16: POKE 14476,02: POKE 14477,169:
    POKE 14478,62
800 POKE 14709,208: POKE 14710,42
810 POKE 14719,96
815 IF PEEK (19988) = 176 THEN POKE 19988,182:
    REM RECONNECT HELP SCREENS
820 POKE 14753,41: POKE 14754,127: POKE 14755,201:
    POKE 14756,96: POKE 14757,176: POKE 14758,208:
    POKE 14759,201
830 POKE 14760,64: POKE 14761,144: POKE 14762,204:
    POKE 14763,41: POKE 14764,63: POKE 14765,176:
    POKE 14766,200
840 PRINT "[D]UNLOCK OBJ.APWRT][E"
850 PRINT "[D]BSAVE OBJ.APWRT][E,A$2300,L$2F5A"
860 PRINT "[D]LOCK OBJ.APWRT][E"
870 TEXT : HOME : A$ = "IT WORKED!": GOSUB 910:
    PRINT : PRINT : PRINT : PRINT : PRINT : END
880 TEXT : HOME :
    A$ = "Will not verify as AWIIE; patch ABORTED":
    GOSUB 910: PRINT : PRINT : PRINT :
    PRINT : PRINT : END
890 GOTO 890
900 TEXT : HOME : CLEAR : END
910 REM
    Noisy screen machine

920 FOR N = 1 TO LEN (A$): PRINT MID$ (A$,N,1);
930 GOSUB 950: REM Clickety clack
940 NEXT N: RETURN

```

**Program A.4—cont.**

```
950  ZZ = PEEK (49200) + PEEK (49200):  
      FOR M = 1 TO 17: NEXT M: RETURN  
960  FOR N = 0 TO 700: NEXT N: RETURN
```

.....

Gotchas: Fixes only the status line. Rare and brief changes  
in the flashing cursor symbol will remain.

In the above listing, [D] stands for "control-D". All  
other brackets are real.

Only the "F" version patch would normally be used by  
the IIc. We have also included an "E" patch for  
possible use by a 64K short "new" IIe.

This program is available ready-to-run on the companion  
diskette

**Program A.5. WPL.SPOKE REARRANGER** *"repairs" many of the oddball codings on the 96 spoke BOLD PS printwheel. Similar routines can be written for other printwheels.*

```

p          WPL.SPOKE REARRANGER
pnd
ppr[L]
ppr      fixing bold PS wheel
ppr
ppr      **** busy - please wait ****
b
f<^~<a
p
p
b
f<!<^<a
p
p
b
f<]<[esc]Z<a
p
p
b
f/</\//a
p watch ut!
p
b
f/[</a
p
p
b
f/|/!/a
p
p
b
f/>/`/a
p
p
b
f/@/[esc]Y/a
p
p
pyd
pqt

```

**Gotchas:** With these corrections, most punctuation gets fixed. The up carat is approximated by a dagger. Lost are both braces, opening and closing single quotes, the tilde and a cents symbol. Gained are two trademarks, a copyright, a degree symbol, a paragraph symbol, a clause symbol, double underline and a dagger. See Table I for proper coding of these new symbols.

While practically all computer listings can be done after conversion, the proportional spacing may cause alignment problems, particularly on hex code dumps. Thus, PS wheels are excellent for people messages but lousy for machine listings. Use TITAN 10 instead.

Note that any WPL modules that precede this routine must respond to the ORIGINAL symbols. WPL modules that follow this one must respond to the REMAPPED symbols. Watch this detail carefully, especially on the underline token and margin settings!

**Program A.5—cont.**

Pairs of [brackets] denote control characters. Single brackets really are brackets.

There are no labels in this particular WPL program, so each and every command must be preceded by a space.

This program is available ready-to-run on the companion disette.

**Program A.6.** *WPLFILE LISTER* file dumper can let you print listings that have buried control commands in them. Most buried commands are replaced with bracketed ASCII characters for "people" viewing.

```

pnd
ppr[L]
ppr          WPL file lister
ppr
pin    WPL source filename ? -----> =$c
pin    Listable object filename -----> =$d
ppr
ppr          **** busy - please wait ****
b
ny
l$c
b
f/[A]/[A]/a
p
b
f/[B]/[B]/a
p
b
f/[C]/[C]/a
p
b
f/[D]/[D]/a
p
b
f/[E]/[E]/a
p
b
f/[F]/[F]/a
p
b
f/[G]/[G]/a
p
b
f/[tab]/[tab]/a
p
b
f/[J]/[J]/a
p
b
f/[K]/[K]/a
p
b
f/[L]/[L]/a
p
b
f/[M]/[M]/a
p
b
f/[O]/[O]/a
p
b
f/[P]/[P]/a
p
b
f/[Q]/[Q]/a
p
b
f/[R]/[R]/a
p
b

```

## Program A.6—cont.

```

f/[S]/[S]/a
p
b
f/[T]/[T]/a
p
b
f/[V]/[V]/a
p
b
f/[W]/[W]/a
p
b
f/[X]/[X]/a
p
b
f/[Y]/[Y]/a
p
b
f/[Z]/[Z]/a
p
b
f/[esc]/[esc]/a
p
b
f/[[]]/[[]]/a
p
b
f/[{}]/[{}]/a
p
b
f/[^]/[^]/a
p
b
f/[_]/[_]/a
p
p
b
s$d
p
b
p
pyd
pqt

```

Gotchas: Pairs of **[shadowed brackets]** substitute for control commands, while all other brackets are real brackets.

There are no labels in this particular WPL program, so each and every command must be preceded by a space.

This program does not repair buried backspace [H] or frontspace [U] control commands. These are quickly fixed by hand, when and if they occur.

Note that the listed program will be easily viewable, printable, or "modemable", but that it will no longer work in its intended manner. To get the listing to actually run, you have to replace the brackets with the control characters they are "standing in" for.

This program is available ready-to-run on the companion diskette.



**Program A.7. WPL.BULLET SHOOTER** inserts line-centered bullets into your text. It automatically substitutes a vertical shifted **BOLD PS** degree symbol for each tilde found in the text.

```
ppr  bullet shooter
p
p  replaces " ~ " with a centered bullet
p  on a diablo 630 BOLD PS wheel.
b
f/  ~  /  [esc]U[esc]D  /a
p
p
pqt
```

---

**Gotchas:** This particular version is intended only for Diablo compatible printers using the BOLD PS wheel.

The hard-to-read line says "find each occurrence of a space followed by a tilde followed by a space; and replace with a space, an escape, an uppercase U, a closing bracket, an escape, an uppercase D, and a space."

For solid bullets, hand fill the bullet with ink.

There are no labels in this particular WPL program, so each and every command must be preceded by a space.

This program is available ready-to-run on the companion disette.

**Program A.8.** *WPL.FLINSERT inserts form letter bodies into an address list. Doing things "backwards" is fast and simple but is limited to 20 letters per run.*

```
p WPL.FLINSERT form letter inserter
b
pasMYLETTER=$a      (name of formletter)
pas***=$b           (insertion marker)
p
a f/$b/
p
pgob
pgobb
b h
f/$b//
y?
l$a
pgoa
bb pqt
.....
```

**Gotchas:** The version shown finds each \*\*\* line in the address list and inserts a formletter called MYLETTER.

Note that MYLETTER must\start\with the body and\end\with the date. The first date needs hand loading.

This is a simple inserter that does not provide a salutation or in-body personalizing. For fancier needs, use the stock formletter routines.

This program is available ready-to-run on the companion disette.

Program A.9. *WPL.CAMERA READY* will slowly doublestrike all of the printed characters for superb final draft print quality. This version is for the Diablo 630.

```

P                                WPL.CAMERA READY
pnd
ppr[L]
ppr  Formating for Diablo 630 "Camera Ready" bold copy
ppr  .....
ppr
ppr                                *** busy - please wait ***
ppr
b
f<><>[esc]O[esc]%<A
P
P
b
f/[esc]O[esc]%././a
P
P
f/[esc]X/[esc]X[esc]O[esc]%/a
P
P
pyd
pqt

```

Gotchas: This code only works on the Diablo 630 printer or on on similar "Diablo Compatible" daisywheel printers.

This module must be run after all carriage returns have been put into the document. Thus, while address labels and charts can be immediately run, longer paragraphs and most text must first be formatted by doing a ".pd8" print to disk. If you fail to do this, only the first line of each paragraph will be double struck; following lines will be hit only once and will look awful.

Best results are gotten with a film ribbon, a metal printwheel, and a machine adjusted to hit a tad on the light side.

Very fine symbols, particularly the BOLD PS trademark, should be hand-bypassed so they are not hit twice.

It is best to use this module immediately before printing. Otherwise, your text files won't be fully compatible with other printers, besides being very hard to read.

Pairs of [brackets] denote control characters.

There are no labels in this particular WPL program, so each and every command must be preceeded by a space.

This program is available ready to run on the companion diskette to this volume.

Program A.10. *This WPL program is a fully automatic "no-frills" formatter for the Diablo 630.*

```

pnd
ppr[L]
pprDiablo 630 "no frills"  Formatter:
ppr.....
ppr
pprPlease enter right margin setting as one SINGLE CHARACTER.
ppr
ppr[.rm50 = 2 .rm60 = < .rm70 = F .rm80 = P .rm90 = Z, etc.]
ppr
pin          Your right margin SINGLE CHARACTER ----->  =$A
ppr
ppr
ppradjusting squashticity
b
psx4 squashticity factor
f<>.dbl<>.dbl>.rm+(x)<
y?
p
e
f<<>.rm-(x)><
y?
p
pprimproving underline
b
f<<.ut><
y?
d f<\< [esc]E<
y?
pgod1
pgod2
dl f<\<[esc]R <
y?
pgod
d2 b
pprefix (.,)
f<[esc]R .<[esc]R.<a
p
p
f<[esc]R ,<[esc]R,<a
p
p
f<.[esc]R<[esc]R.<a
p
p
f<,[esc]R<[esc]R,<a
p
p
f<.><.><a
p
p
pprefixing underline bug
e
e u
f<[esc]E<
p
pgoe1
pgoe2
el f<><>[esc]\<
y?
f<[esc]E<
p
pgoe

```

## Program A.10—cont.

```

e2 p
pprfix bold PS wheel
b
f<![^<a
p
p
b
f<][esc]z<a
p
p
b
f/</\//a
p watch ut!
p
b
f/[</a
p
p
b
f/|/!/a
p
p
b
f/>/`/a
p
p
b
f/@/[esc]Y/a
p
p
pprtightening spacing
b
f<>.dbl<
p
pgob
pgob3
b h
h
f<>>>>>[esc]U<
y?
pgob1
pgob3
bl f<>>>>>[esc]D<
y?
pgob2
pgob3
b2 f<>>>><
p
pgob
b3 p
pprsetting body microjustify
b
f<>.dbl<
p
f<>.fj>>[esc]X[esc]N[esc][tab]z[esc][tab]$A[esc]O[esc]
[tab]z [esc][tab][A][esc]/>.lj >[esc]M<A
p
pprshadowing titles
b
f<>.dbl<
p
pgoc
pgocl
c h
h

```

## Program A.10—cont.

```

f<>.cj><>.cj>[esc]\[esc]X[esc]W[esc][Q][B][esc]% <
y?
p
f<><[esc]N[esc]P><
y?
h
f<>.cj<
p
pgoc
cl p
pprefixing paragraph ends
b
f<>.dbl<
p
f<.><[esc]X.[esc]7[esc]/>[esc]M<a
p
b
f<>.dbl<
p
f<^><[esc]X^[esc]7[esc]/>[esc]M<a
p !=^ bold ps
b
f#?%#[esc]X?[esc]7[esc]/%[esc]M#a
p
b
f<'><[esc]X'[esc]7[esc]/>[esc]M<a
p >=' bold ps
pin[G][G][G]--- detail work filename? ---> =$d
psz+1 for wpl supervisor
pas$d =$d
pcs/ /$d/
pgog
pdo$d
g pqt

```

.....

Gotchas: Pairs of brackets mean control commands. [esc] = escape key, [L] means "<ctrl>L", etc. Note that any isolated brackets really are isolated brackets.

The indented line in the "body justify" section is a continuation of the previous line and must be entered without intervening spaces or returns. Note that each and every line must begin with either a label or a space.

This WPL program assumes a Diablo 630 printer with an enhanced HP205 board having full word processing features. The program MUST be customized if any other printer is to be used. Certain features may not be available on other printers.

This program is available ready to run on the companion diskette.



Program A.11. Apple DMP formatting glossary with tutorial.

```

?
?   Applewriter IIe/Apple DMP formatting glossary "AGLOSS"
?
A[V] [esc] [K] [V]
a[V] [esc] [tab] [V]
B[V] [esc] K[V]
b[V] [esc] M[V]
C[V] [L] [V]
c[V] [esc] 2[V]
D[V] [esc] N[V]
d[V] [esc] G[V]
E[V] [esc] 0[V]
e[V] [esc] 9[V]
F[V] [esc] F[V]
f[V] [esc] F66[V]
G[V] [esc] 3[V]
g[V] [esc] 4[V]
H[V] ^ [V]
h[V] _ [V]
I[V] [esc] [Z] [E] [V]
i[V] [esc] : [V]
J[V] [E] [V]
j[V] [F] [V]
K[V] [esc] [E] [V]
k[V] [esc] [F] [V]
L[V] [esc] 1[V]
l[V] [esc] [H] [V]
M[V] [esc] H[V]
m[V] [esc] V[V]
N[V] [esc] + [V]
n[V] [esc] - [V]
O[V] [esc] D[V]
o[V] [esc] U[V]
P[V] [esc] $ [V]
p[V] [esc] % [V]
Q[V] [esc] S[V]
q[V] [esc] T[V]
R[V] [esc] < [V]
r[V] [esc] > [V]
S[V] [esc] Q[V]
s[V] [esc] R[V]
T[V] [esc] 1 [V]
t[V] [esc] 8 [V]
U[V] [esc] I[V]
u[V] [esc] J[V]
V[V] [esc] [J] [V]
v[V] [J] [V]
W[V] [esc] [V]
w[V] [esc] / [V]
X[V] [esc] [Z] I[V]
x[V] [esc] ] P[V]
y[V] [V]
y[V] [V]

```

#### Apple DMP Open-Apple Formatting Commands:

```

.....
.....

```

```

(A) absolute Vmove* (H) vert VMI set*
(O) offset hl up    (V) vertical up
(a) absolute Hmove* (h) horiz HMI set*
(o) offset hl down  (v) vertical down

```

## Program A.11—cont.

(B) bold print on*	(I) inquire request
(P) prop. space on	(W) wheel spoke 004
(b) bold print off	(i) inquire reply*
(p) prop. space off	(w) wheel spoke 002
(C) cause formfeed	(J) jump ext ASCII
(Q) quit printing	(X) reset hard
(c) clear all tabs	(j) jump std ASCII
(q) resume printing	(x) reset soft
(D) doublestrike on	(K) kustom prog on*
(R) reverse <- on	(Y) yourstuff on
(d) double graphics	(k) kustom prog off
(r) reverse <- off	(y) yourstuff off
(E) west margin set	(L) language pick*
(S) shadowprint on	(Z) tutorial+ save
(e) east margin set	(l) little backspce
(s) shadowprint off	(z) tutorial only
(F) formlength set*	(M) move Vrelative*
(T) tab horiz set	
(f) formlength 66	(m) move Hrelative*
(t) tab horiz clear	
(G) graphics on	(N) north margin
(U) underline on	
(g) graphics off	(n) south margin
(u) underline off	

Capital letter is "on", "above", "vertical", or "right".

\* - follow with data values (See printer manual or help card)

```
Z[P]nd] [Q]FAGLOSS] [L]AGLOSS<>    <ard>>><\\] [P]yd]
z[L]AGLOSS<>    <ard>>><\\]
```

Gotchas: Pairs of brackets mean control commands. [esc] = escape key, [Q] means "<ctrl>Q", etc. Note that any isolated brackets really are isoated brackets.

The line preceeding the tutorial screen must have four spaces on it. "Z" and "z" selections must NOT preceed the tutorial or they will find themselves.

Eighty column tutorial text lines have been split. Entries (A), (H), (O), and (V) all go on one line. The dot row is also one continuous line.

This program is available ready to run on the companion diskette.

Program A.12. *Diablo 630 formatting glossary with tutorial.*

```

?  Applewriter IIe/Diablo 630 Formatting Glossary "DGLOSS"
?
A[V] [esc] [K] [V]
a[V] [esc] [U] [V]
B[V] [esc] O[V]
b[V] [esc] & [V]
C[V] [esc] = [V]
c[V] [esc] X[V]
D[V] [esc] $ [V]
d[V] [esc] X[V]
E[V] [esc] O[V]
e[V] [esc] 9[V]
F[V] [esc] A[V]
f[V] [esc] B[V]
G[V] [esc] 3[V]
g[V] [esc] 4[V]
H[V] [K] [V]
h[V] [tab] [V]
I[V] [esc] & [V]
i[V] [esc] N[V]
J[V] [esc] M[V]
j[V] [esc] X[V]
K[V] [esc] [Q] [V]
k[V] [esc] X[V]
L[V] [esc] [L] [V]
l[V] [esc] [R] [V]
M[V] [esc] ^ [V]
m[V] [esc] _ [V]
N[V] [esc] T[V]
n[V] [esc] S[V]
O[V] [esc] D[V]
o[V] [esc] U[V]
P[V] [esc] P[V]
p[V] [esc] Q[V]
Q[V] [esc] 7[V]
q[V] [esc] X[V]
R[V] [esc] / [V]
r[V] [esc] \ [V]
S[V] [esc] W[V]
s[V] [esc] & [V]
T[V] [esc] - [V]
t[V] [esc] 1[V]
U[V] [esc] E[V]
u[V] [esc] R[V]
V[V] [esc] [J] [V]
v[V] [J] [V]
W[V] [esc] Z[V]
w[V] [esc] Y[V]
X[V] [esc] 2[V]
x[V] [esc] S[V]
Y[V] [V]
y[V] [V]
^ [V] [L] [V]
< [V] [R] [V]
, [V] [esc] Q[esc] 1[R] [tab] [V]
. [V] [esc] P[V]

```

Diablo 630 Open-Apple Formatting Commands:

.....

(A) absolute VTAB\*    (R) vertical tab

## Program A.12—cont.

(O) offset hl up	(V) vertical up
(a) absolute HTAB*	(h) horizontal tab
(o) offset hl down	(v) vertical down
(B) bold print on	(I) improve quality
(P) proportional on	(W) wheel spoke \$7F
(b) bold print off	(i) normal quality
(p) prop. space off	(w) wheel spoke \$20
(C) centering on	(J) justify on
(Q) quit printing	(X) clear tabs
(c) centering off	(j) justify off
(q) quit wp modes	(x) clear EMI
(D) dash hyphen on	(K) kerning set *
(R) reverse <- on	(Y) yourstuff on
(d) dash hyphen off	(k) kill v margin
(r) reverse <- off	(y) yourstuff off
(E) west margin set	(L) lines/page set*
(S) shadow on	(Z) tutorial + copy
(e) east margin set	(l) little backspce
(s) shadow off	(z) tutorial only
(F) funny ribbon on	(M) motion VMI **
(T) tab vert. set	(^) formfeed
(f) black ribbon on	(m) motion EMI **
(t) tab horiz. set	(<) backspace
(G) graphics on	(N) north margin
(U) underline on	(,) dots start
(g) graphics off	(n) south margin
(u) underline off	(.) dots end

Capital letter is "on", "more", "above", "vertical", or "right".

\* - follow with ASCII value  
 \*\* - follow with ASCII value-1

```
Z[P]nd][Q]FDGLOSS][L]DGLOSS<>    <-1>><\][P]yd]
z[L]DGLOSS<>    <-1>><\]
```

.....

Gotchas: Pairs of brackets mean control commands. [esc] = escape key, [Q] means "<ctrl>Q", etc. Note that any isolated brackets really are isolated brackets.

The line preceeding the tutorial screen must have four spaces on it. "Z" and "z" selections must NOT preceed the tutorial or they will find themselves.

Eighty column tutorial text lines have been split. Entries (A), (H), (O), and (V) all go on one line. The dot row is also one continuous line.

This program is available ready to run on the companion diskette.

Program A.13. Epson MX80 formatting glossary with tutorial.

```

?   Applewriter IIe/Epson MX80 formatting glossary "EGLOSS"
?
A[V] [esc] # [V]
a[V] [esc] > [V]
B[V] [esc] G [V]
b[V] [esc] H [V]
C[V] [esc] [O] [V]
c[V] [esc] [S] [V]
D[V] [esc] [N] [V]
d[V] [esc] [T] [V]
E[V] [esc] E [V]
e[V] [esc] F [V]
F[V] [esc] C [V]
f[V] [esc] C [ @ ] [V]
G[V] [esc] L [V]
g[V] [esc] K [V]
H[V] [esc] J [V]
h[V] [esc] 2 [V]
I[V] [esc] 4 [V]
i[V] [esc] 5 [V]
J[V] [esc] N [V]
j[V] [esc] O [V]
K[V] [V]
k[V] [V]
L[V] [esc] = [esc] [J] [V]
l[V] [esc] > [esc] [J] [V]
M[V] [V]
m[V] [V]
N[V] [G] [V]
n[V] [G] [V]
O[V] [esc] U [ @ ] [V]
o[V] [esc] U1 [V]
P[V] [esc] 9 [V]
p[V] [esc] 8 [V]
Q[V] [V]
q[V] [V]
R[V] [esc] = [V]
r[V] [esc] > [V]
S[V] [esc] S [ @ ] [V]
s[V] [esc] S1 [V]
T[V] D [V]
t[V] D [V]
U[V] [esc] -1 [V]
u[V] [esc] - [ @ ] [V]
V[V] [esc] 1 [V]
v[V] [esc] 0 [V]
W[V] [esc] Q [V]
w[V] [esc] Q [V]
X[V] [esc] @ [V]
x[V] [esc] T [V]
Y[V] [V]
y[V] [V]
^ [V] [L] [V]
< [V] [H] [V]

```

Epson MX-80 Open-Apple Formatting Commands:

.....  
 .....

(A) ascii b8 as is    (H) height custom \*  
 (O) two way print    (V) VERY tight 7/72

## Program A.13—cont.

(a) ascii b8 one	(h) height normal
(o) one way print	(v) very tight 1/8
(B) bold print on	(I) italics on
(P) paperout sense	(W) width column *
(b) bold print off	(i) italics off
(p) paperout ignore	(w) width column *
(C) compressed on	(J) jump perf on *
(Q) (spare)	(X) off all modes
(c) compressed off	(j) jump perf off
(q) (spare)	(x) off sub/super
(D) doublewide on	(K) (spare)
(R) reset B8 = 1	(Y) yourstuff on
(d) doublewide off	(k) (spare)
(r) reset B8 = 0	(y) yourstuff off
(E) emphasized on	(L) linefeed w/rst
(S) superscript on	(Z) tutorial + save
(e) emphasized off	(l) linefeed only
(s) subscript on	(z) tutorial only
(F) fmlngth lines *	(M) (spare)
(T) tab set *	(^) formfeed
(f) fmlngth inches *	(m) (spare)
(t) tab set *	(<) backspace
(G) graphics 960 *	(N) noisy bell
(U) underline on	
(g) graphics 480 *	(n) noisy bell
(u) underline off	

Capital letter is "on", "yes", "above", or "more".  
For full features, the AWIIE NULL patch is needed.

\* - follow with data value(s). (See Epson user manual for details.)

```
Z[Q]FEGLOSS[L]EGLOSS<>    <ls.)>><\]
z[L]EGLOSS<>    <ls.)>><\]
```

Gotchas: Pairs of brackets mean control commands. [esc] = escape key, [Q] means "<ctrl>Q", etc. Note that any isolated brackets really are isolated brackets.

The line preceeding the tutorial screen must have four spaces on it. "Z" and "z" selections must NOT preceed the tutorial or they will find themselves.

The NULLIFIER patch of chapter one is needed for underlining and superscripts on older printers.

Eighty column tutorial text lines have been split. Entries (A), (H), (O), and (V) all go on one line. The dot row is also one continuous line, as is the line starting with "\* - Follow"

This program is available ready to run on the companion diskette.



Program A.14. *Imagewriter formatting glossary with tutorial.*

```

?
?   Applewriter IIe/Imagewriter formatting glossary "IGLOSS"
?
A[V] [esc]O[V]
a[V] [esc]o[V]
B[V] [esc]! [V]
b[V] [esc]" [V]
C[V] [L] + [V]
c[V] [esc] - [V]
D[V] [esc] [H] [V]
d[V] [esc] R [V]
E[V] [esc]O[V]
e[V] [esc]L [V]
F[V] { } A@ [V]
f[V] [L] {V}
G[V] [esc]V[V]
g[V] [esc]G[V]
H[V] [N] [V]
h[V] [O] [V]
I[V] [esc]G[V]
i[V] [ ] [V]
J[V] [esc]Z@ [V]
j[V] [esc]D@ [V]
K[V] [esc]s [V]
k[V] [esc] [V]
L[V] [esc]D [V]
l[V] [esc]z [V]
M[V] [esc]Q[V]
m[V] [esc]q[V]
N[V] [esc]e[V]
n[V] [esc]E[V]
O[V] [esc]N[V]
o[V] [esc]n[V]
P[V] [esc]p[V]
p[V] [esc]P[V]
Q[V] [esc]' [V]
q[V] [esc]$ [V]
R[V] [esc]< [V]
r[V] [esc]> [V]
S[V] [esc]B[V]
s[V] [esc]A[V]
T[V] [esc] ([V]
t[V] [esc]) [V]
U[V] [esc]X[V]
u[V] [esc]Y[V]
V[V] [esc]r[V]
v[V] [esc]f[V]
W[V] [esc]Z [V]
w[V] [esc]D [V]
X[V] c[V]
x[V] [X] [V]
Y[V] [V]
y[V] [V]

```

Imagewriter Open-Apple Formatting Commands:

.....  
 .....

(A) alarm on	(H) headline on
(O) ordinary 10	(V) vertical up
(a) alarm off	(h) headline off
(o) obese wide 9	(v) vertical down

## Program A.14—cont.

(B) bold print on	(I) insert columns*
(P) prop space pica	(W) will use b8
(b) bold print off	(i) insert blanks*
(p) prop spce elite	(w) will not use b8
(C) custom 16 wide	(J) jump on CR only
(Q) quit normal	(X) cancel all
(c) custom 8 wide	(j) jump on all
(q) quit custom	(x) cancel text
(D) doublewhap*	(K) kern spaces*
(R) reverse <- on	(Y) yourstuff on
(d) duplicate char*	(k) kern characters*
(r) reverse <- off	(y) yourstuff off
(E) erase all tabs	(L) linefeed enable
(S) space 8 LPI	(Z) tutorial + save
(e) east mrgin set*	(l) linefeed dsable
(s) space 6 LPI	(z) tutorial only
(F) form TOF set	(M) mighty tight 17
(T) tab set	
(f) formfeed	(m) midi tight 15
(t) tab clear	
(G) graphics X8 on	(N) near tight 13
(U) underline on	
(g) graphics X1 on	(n) normal 10
(u) underline off	

Capital letter is "on", "above", "vertical", or "right".

\* - follow with data values (See printer manual or help card)

```
Z[P]nd] [Q]FIGLOSS] [L]IGLOSS<>    <ard)>><\] [P]yd]
z[L]IGLOSS<>    <ard)>><\]
```

Gotchas: Pairs of brackets mean control commands. [esc] = escape key, [Q] means "<ctrl>Q", etc. Note that any isolated brackets really are isoated brackets.

The line preceeding the tutorial screen must have four spaces on it. "Z" and "z" selections must NOT preceed the tutorial or they will find themselves.

The NULLIFIER patch is needed for certain seldom-used commands. See chapter one.

Eighty column tutorial text lines have been split. Entries (A), (H), (O), and (V) all go on one line. The dot row is also one continuous line.

This program is available ready to run on the companion diskette.



---

# B

---

## Machine Language Patches

### How to patch Applewriter IIe.

1. The patches that follow are totally unofficial and are in no way supported by nor approved of by either Apple Computer or the original program author.
2. Patches 6.1 - 6.7 are intended only for the DOS 3.3 versions of Applewriter IIe. There are two versions of the program, the "E" version, or OBJ.APWRT][E for use in a 64K Apple IIe without extended memory, and the "F" version which is named OBJ.APWRT][F for use in a 128K Apple IIe that has extended memory. The booting code OBJ.BOOT automatically picks the correct program after testing for auxiliary IIe memory.  
  
Use "E" patches ONLY on OBJ.APWRT][E code. Use "F" patches ONLY on OBJ.APWRT][F code.
3. Patch only your third or higher backup copy. Do NOT patch either the stock diskette or its factory backup.
4. Cold boot stock DOS 3.3e from your system master diskette.
5. To make an "E" patch:
  - (a) BLOAD OBJ.APWRT][E, A\$2300
  - (b) CALL -151 to get into the system monitor
  - (c) Verify the patch area. DO NOT CONTINUE UNLESS YOUR CODE EXACTLY AGREES!
  - (d) Enter the patch code.
  - (e) Check the patch code for correct entry.
  - (f) UNLOCK OBJ.APWRT][E
  - (g) BSAVE OBJ.APWRT][E, A\$2300, L\$2F58
  - (h) LOCK OBJ.APWRT][E
  - (i) Test your patch using non-valuable text files.
6. To make an "F" patch:
  - (a) BLOAD OBJ.APWRT][F, A\$2300
  - (b) CALL -151 to get into the system monitor
  - (c) Verify the patch area. DO NOT CONTINUE UNLESS YOUR CODE EXACTLY AGREES!
  - (d) Enter the patch code.
  - (e) Check the patch code for correct entry.
  - (f) UNLOCK OBJ.APWRT ][F
  - (g) BSAVE OBJ.APWRT][F, A\$2300, L\$30D2
  - (h) LOCK OBJ.APWRT][F
  - (i) Test your patch using non-valuable text files.

**Patch B.1. The NULLIFIER (AWHe).**

**What it does:** Allows imbedded NULL commands for such things as superscript and underline on older Epson printers.

After this mod, NULLs may be imbedded into your textfiles by using a [V][@][V] command.

**How it works:** Branches that eliminate entry of \$80 NULLs are shortened so they go nowhere.

**Side effects:** Obscure use of DELETE key is lost, but you should never be using this key anyway. Do not use in 40 column (no-card) mode. Do not use a NULL as a character in a WPL label.

**The "E" patch:**

1. Verify 26E1- C9 80 F0 0F

Patch 26E4: 00

2. Verify 48BA- C9 80 F0 D6

Patch 48BD: 00

**The "F" patch:**

1. Verify 2781- C9 80 F0 15

Patch 2784: 00

2. Verify 4A33- C9 80 F0 D6

Patch 4A36: 00



Patch B.2. *The STRETCHIFIER (AWIIe).*

**What it does:** Eliminates the "short line" bug when imbedded printer commands are counted as part of a fill justified line. Each imbedded [esc] command in a line lengthens only that line by TWO counts.

**How it works:** Intercepts line length counter routine and then lengthens line by two counts for each imbedded [esc] command. Then knocks off any "unused" counts bypassed by word wraparound.

**Side effects:** Exact fix results only for two-character imbedded sequences. For three-character sequences, extra counts can be "banked", by using [esc][esc] to bank a pair of counts, or [esc][esc][2] to bank a single count. Such banking can be built into the glossary that holds the imbedding commands. To make room for this code, the unneeded AWIIe "volume verify" routine is shortened to a default return command.

The "E" patch:

```

1. Verify 397F- 20 EB 46 A9 01
   Patch  397F: 60
           3980: 99 00 16 C9 9B D0 04 E6
           3988: D3 E6 D3 60 C4 DC F0 0E
           3990: B9 00 16 C9 9B D0 04 C6
           3998: D3 C6 D3 88 D0 EE 4C FC
           39A0: 45

2. Verify 43F4- 99 00 16
   Patch  43F4: 20 80 39

3. Verify 441B- 4C FC 45
   Patch  441B: 4C 8C 39

4. Verify 4E12- AC D3 B0
   Patch  4E14: B6

```

The "F" patch:

```

1. Verify 3AD6- 20 64 48 A9 01
   Patch  3AD6: 60 99
           3AD8: 00 16 C9 9B D0 04 E6 D3
           3AE0: E6 D3 60 C4 DC F0 0E B9
           3AE8: 00 16 C9 9B D0 04 C6 D3
           3AF0: C6 D3 88 D0 EE 4C 75 47

2. Verify 456B- 99 00 16
   Patch  456B: 20 D7 3A

3. Verify 4592- 4C 75 47
   Patch  4592: 4C E3 3A

4. Verify 4F8B- AC D3 B0
   Patch  4F8D: B6

```

**Patch B.3. The CURSIFIER (AWIIe).**

**What it does:** Puts the cursed character into the WPL \$D string on a [Q]-K command. This greatly speeds up WPL programs that need character-by-character logic.

**How it works:** Pointer to flashing cursor is decremented. Pointed value is then put into the \$D string as its first character. The second \$D string character is forced to \$00. The cursor pointer is then incremented back to its original position.

**Side effects:** The normally unused "Quit" command is no longer available. The \$D string must have nothing useful in it before doing a [Q]-K.

**The "E" patch:**

1. Verify 2BD1- A0 00 B9 19

Patch 2BD1: 20 D6 28 B1 84 8D 40  
2BD8: 18 A9 00 8D 41 18 4C 8C  
2BE0: 28

2. Verify 5047- D1 F5 E9 F4

Patch 5047: C3  
5048: F5 F2 F3 EF F2 A0 AD AD  
5050: BE A0 A4 C4 A0 A0 A0 A0

**The "F" patch:**

1. Verify 2CA4- A0 00 B9 92

Patch 2CA4: 20 9E 29 20  
2CA8: 32 01 8D 40 18 A9 00 8D  
2CB0: 41 18 4C 54 29

2. Verify 51C0- D1 F5 E9 F4

Patch 51C0: C3 F5 F2 F3 EF F2 A0 AD  
51C8: AD BE A0 A4 C4 A0 A0 A0  
51D0: A0

**Patch B.4. The PATCHIFIER (AWIIe).**

**What it does:** Allows you to modify or extend any part of Applewriter at any time for any reason, by using a "Blood Patch" command under [O]-C. Also adds a rudimentary PEEK and POKE to WPL.

**How it works:** The seldom used "Verify File" command is changed so it will load any binary file of your choice. Only the actual DOS 3.3e command is altered.

**Side effects:** Verifying DOS files becomes much harder. Matching patch to version is absolutely essential. Careless use of this powerful command can destroy everything within a one mile radius of your IIe, so BE CAREFUL!

The "E" patch:

```
1. Verify 4D80- D6 E5 F2 E9
Patch 4D80: C2 EC EF E1 E4 A0 D0 E1
      4D88: F4 E3 E8
```

The "F" patch:

```
1. Verify 4EF9- D6 E5 F2 E9
Patch 4EF9: C2 EC EF E1 E4 A0 D0
      4F00: E1 F4 E3 E8
```

**Patch B.5. The LINKIFIER (AWIIe).**

**What it does:** Executes your own custom machine language module when you or WPL do a [Q]-H.

**How it works:** Diverts the unneeded [Q]-H function to jump to your code as a subroutine.

**Side effects:** Integrity and safety of entire program depends on your custom modules. Safe areas for custom code are main mamory \$5258-52FF for the "E" version or \$53D1-\$BEFF for the "F" version.

**The "E" patch:**

1. Verify 508F: 36 31

Patch 508F: Module start low address  
5090: Module start high address

2. Verify 4FEA: D4 EF E7 E7 EC E5 A0 C4

Patch 4FEA: Module name using exactly  
24 ASCII characters

**The "F" patch:**

1. Verify 5208: 5B 32

Patch 5208: Module start low address  
5209: Module start high address

2. Verify 5163: D4 EF E7 E7 EC E5 A0 C4

Patch 5163: Module name using exactly  
24 ASCII characters

Patch B.6. *The CLARIFIER (AWIIe).*

**What it does:** Eliminates trashing of the 2C status line display.

**How it works:** Inverse upper case status characters are remapped so they do not conflict with the mousetext area in the 2c character generator.

**Side effects:** Remaining portion of unused "volume verify" code is used by this patch. Repairs only the status line. Rare and brief changes in the flashing cursor symbol will remain.

## The "E" patch:

1. Verify 3888- BC 24 CF 10 02 A9 BE  
Patch 3888: 3C 24 CF 10 02 A9 3E
2. Verify 3975- 29 7F 84  
Patch 3975: D0 2A
3. Verify 397F- 20 -or- 60  
Patch 397F: 60
4. Verify 39A1- 39 C5 82 D0  
Patch 39A1: 29 7F C9 60 B0 D0 C9  
39A8: 40 90 CC 29 3F B0 C8
5. Verify 4E12- AC D3 B0  
Patch 4E14: B6

## The "F" patch:

1. Verify 39DF- BC 24 CF 10 02 A9 BE  
Patch 39DF: 3C 24 CF 10 02 A9 3E
2. Verify 3ACC- 29 7F 84  
Patch 3ACC: D0 2A
3. Verify 3AD6- 20 -or- 60  
Patch 3AD6: 60
4. Verify 3AF8- 3B C5 82 D0  
Patch 3AF8: 29 7F C9 60 B0 D0 C9 40  
3B00: 90 CC 29 3F B0 C8
5. Verify 4F8B- AC D3 B0  
Patch 4F8D: B6

**Patch B.7. The RESTORIFIER (AWIIe).**

**What it does:** Restores the help screens should the "volume verify" routine be diverted for other patches.

**How it works:** The generic slot number in the help screen code is overwritten to force slot six.

**Side effects:** The disk drives must now be in slot six for the help screens to work.

**The "E" patch:**

1. Verify 4E12- AC D3 B0

Patch 4E14: B6

**The "F" patch:**

1. Verify 4F8B- AC D3 B0

Patch 4F8D: B6



**Patch B.8. The PREFIXIFIER (ProDOS 2.0).**

**For:** ProDOS Applewriter 2.0, versions AWB.SYS (for 40 column IIc), AWC.SYS (for 64K IIe, or AWD.SYS (80 column IIc or 128K IIe).

**What it does:** Automatically sets the prefix to whatever is in drive two on bootup.

**How it works:** Does a "set prefix, d2" as part of STARTUP program.

**Gotchas:** Has to sit in drive one during bootup. A non-fatal error message results if you have nothing in drive two.

**The patch:**

1. Make a third or higher backup copy of ProDOS Applewriter 2.0, using the filer.
2. Open a new Applewriter text file.
3. Type "<space> oh,d2] <return>"
4. Save file under name STARTUP to your new backup copy.

**NOTE:** If you are using another STARTUP program, just add the line in step #3 to your existing WPL routine.

**Patch B.9. The AIOIFIER (ProDOS 2.0).**

**For:** ProDOS Applewriter 2.0, version AWD.SYS only.  
AWD.SYS runs only on the 80 column IIc or the 128K IIfx.

**What it does:** Eliminates swallowed imbedded print commands if an attempt is made to use the AIO card as a serial printer interface.

**How it works:** Convinces the card that it is not to echo screen video, nor output an extra space when at the left margin.

**Gotchas:** Uses memory locations from \$5FF0 through \$5FFB, May do very bizarre things to other printer cards. Conflicts with GRAPPLIFIER patch. Works only in slot one.

**The patch:**

1. Make a third or higher backup copy of ProDOS Applewriter 2.0, using the filer.
2. Get into /BASIC.SYS. CALL -151 to get into monitor.
3. BLOAD AWD.SYS, A\$2000, E\$6020, TSYS,D2
4. [A] Verify: \$4CA0- 6C 9E 00  
[B] Change: \$4CA0: 4C F0 5F
5. [A] Verify: \$5FF0- 1A 1A 1A 1A 1A . . .  
[B] Change: \$5FF0: A0 FF 8C F9 07 C8 8C 79  
\$5FF8: 07 6C 9E 00
6. [A] Verify: \$4CA0-4CA2 per above  
[B] Verify: \$5FF0-6001 per above
7. BSAVE AWD.SYS, A\$2000, E\$6020, TSYS.

**Patch B.10. The GRAPPLIFIER (ProDOS 2.0).**

**For:** ProDOS Applewriter 2.0, version AWD.SYS only. AWD.SYS runs only on the 80 column IIc or the 128K IIe.

**What it does:** Eliminates random bursts of spaces whenever an attempt is made to use the Grappler card as a parallel printer interface.

**How it works:** Performs a frontal lobotomy on the card just before each character is printed. Does this by defeating screen echo, stopping all modes, setting length to \$00, left margin and unused memory location \$24 to \$01, and internal cursor to \$02.

**Gotchas:** Uses memory locations from \$5FF0 through \$6002, thus lengthening the program by two bytes. May do very bizarre things to other parallel cards. Conflicts with AIOIFIER patch. Works only in slot one.

**The patch:**

1. Make a third or higher backup copy of ProDOS Applewriter 2.0, using the filer.
2. Get into /BASIC.SYS. CALL -151 to get into monitor.
3. BLOAD AWD.SYS, A\$2000, E\$6020, TSYS,D2
4. [A] Verify: \$4CA0- 6C 9E 00  
[B] Change: \$4CA0: 4C F0 5F
5. [A] Verify: \$5FF0- 1A 1A 1A 1A 1A . . .  
[B] Change: \$5FF0: A0 00 8C 79 04 8C F9 07  
\$5FF8: C8 84 24 C8 8C F9 04 6C  
\$6000: 9E 00
6. [A] Verify: \$4CA0-4CA2 per above  
[B] Verify: \$5FF0-6001 per above
7. BSAVE AWD.SYS, A\$2000, E\$6020, TSYS.

**Patch B.11. The BOOTIFIER (ProDOS 2.0).**

**For:** ProDOS Applewriter 2.0, version AWD.SYS only. AWD.SYS runs only on the 80 column IIc or the 128K IIe.

**What it does:** Gives a slightly faster bootup by eliminating the first screen display and the key prompt.

**How it works:** Bypasses the first display screen and key prompt by replacing a subroutine call with NOP's.

**Gotchas:** Monumentally uninformative to beginning or casual users. Best used with a controlling STARTUP program in drive one.

**The patch:**

1. Make a third or higher backup copy of ProDOS Applewriter 2.0, using the filer.
2. Get into /BASIC.SYS. CALL -151 to get into monitor.
3. BLOAD AWD.SYS, A\$2000, E\$6020, TSYS,D2
4. [A] Verify: \$20AB- 20 E5 21  
[B] Change: \$20AB: EA EA EA
5. [A] Verify: \$20AB-20AD per above.
6. BSAVE AWD.SYS, A\$2000, E\$6020, TSYS.

**Patch B.12. The NULLIFIER (ProDOS 2.0).**

**For:** ProDOS Applewriter 2.0, version AWD.SYS only.  
AWD.SYS runs only on the 80 column IIC or the  
128K IIC.

**What it does:** Lets you redefine a substitute character for  
NULL. The current US user separator [\_] may  
conflict with certain modems and some daisywheel  
HMI commands.

**How it works:** Substitutes the desired character by changing  
the operand of an immediate compare instruction.

**Gotchas:** Patch is needed only if you need the US user  
separator [\_] for itself, rather than a NULL  
substitution sit-in.

**The patch:**

1. Make a third or higher backup copy of ProDOS  
Applewriter 2.0, using the filer.
2. Get into /BASIC.SYS. CALL -151 to get into monitor.
3. BLOAD AWD.SYS, A\$2000, E\$6020, TSYS,D2
4. Verify \$4C36 as a \$C9 and \$4C37 as the current  
NULL substitution character.
5. Change \$4C37 to the new NULL substitution character.  
Use [\_] user separator US as default, or \$00 for no  
NULLs at all.
6. [A] Verify: \$4C36-4C37 per above.
7. BSAVE AWD.SYS, A\$2000, E\$6020, TSYS.

**Patch B.13. The GLOSSIFIER (ProDOS 2.0).**

**For:** ProDOS Applewriter 2.0, version AWD.SYS only.  
AWD.SYS runs only on the 80 column IIC or the 128K IIE.

**What it does:** Eliminates the potential ability to overwrite the glossary and destroy the program. Corrects a code byte that is just plain wrong.

**How it works:** Before entering a character from the keyboard into the glossary, a check is made to make sure there is at least enough room for an ending carriage return and \$00 marker. If not enough room, an error message results. The correct maximum address is \$0FFD, not \$67FD.

**Gotchas:** None known.

**The patch:**

1. Make a third or higher backup copy of ProDOS Applewriter 2.0, using the filer.
2. Get into /BASIC.SYS. CALL -151 to get into monitor.
3. BLOAD AWD.SYS, A\$2000, E\$6020, TSYS,D2
4. [A] Verify: \$2B56- 67  
[B] Change: \$2B56: 0F
5. [A] verify: \$2B56 per above.
6. BSAVE AWD.SYS, A\$2000, E\$6020, TSYS.



**Patch B.14. The CREEPIFIER (ProDOS 2.0).**

**For:** ProDOS Applewriter 2.0, version AWD.SYS only.  
AWD.SYS runs only on the 80 column IIc or the 128K IIe.

**What it does:** Eliminates an extra space at the end of all top and bottom lines, thus eliminating a potential page creep problem if user sets both program and printer right margins to 80 characters, and suppresses first page headers or footers.

**How it works:** Enters a "print space and carriage return" sub in the middle, so it only prints a return.

**Gotchas:** None known. It is utterly amazing how many calls have been made on this. The alternate and obvious cure is to use RM78 as a print constant.

**The patch:**

1. Make a third or higher backup copy of ProDOS Applewriter 2.0, using the filer.
2. Get into /BASIC.SYS. CALL -151 to get into monitor.
3. BLOAD AWD.SYS, A\$2000, E\$6020, TSYS,D2
4. [A] Verify: \$47ED- 20 53 46  
[B] Change: \$47ED: 20 58 46
5. [A] Verify: \$47ED-47EF per above
6. BSAVE AWD.SYS, A\$2000, E\$6020, TSYS.

Patch B.15. *The SCRUNCHIFIER (ProDOS 2.0).*

**For:** ProDOS Applewriter 2.0, version AWD.SYS only.  
AWD.SYS runs only on the 80 column IIc or the 128K IIe.

**What it does:** Shortens the DOS options menu to four lines, leaving more of a previous catalog on screen. Also frees up room in code for other patches.

**How it works:** Rewrites the ASCII image of the DOS screen menu so it is much shorter.

**Gotchas:** Style inconsistent with other menus. Spelling of "Subdirectory" uses a convention only seen in the upper reaches of Marijilda Canyon in the Pinaleno Mountains. This patch must be made if the STRETCHIFIER and the CURSIFIER patches are to be used.

**The patch:**

1. Make a third or higher backup copy of ProDOS Applewriter 2.0, using the filer.
2. Get into /BASIC.SYS. CALL -151 to get into monitor.
3. BLOAD AWD.SYS, A\$2000, E\$6020, TSYS,D2
4. [A] Verify: \$591C- 20 20 20 50 72 . . .

[B] Change:

\$591C:	50 72 6F 44
\$5920:	6F 73 3A 20 28 41 29 20
\$5928:	43 61 74 61 6C 6F 67 20
\$5930:	20 28 43 29 20 4C 6F 63
\$5938:	6B 20 20 20 20 28 45 29
\$5940:	20 44 65 6C 65 74 65 20
\$5948:	20 20 28 47 29 20 53 75
\$5950:	62 64 72 63 74 79 20 28
\$5958:	49 29 20 46 6F 72 6D 61
\$5960:	74 20 20 20 0D 20 20 20
\$5968:	20 20 20 20 20 28 42 29
\$5970:	20 52 65 6E 61 6D 65 20
\$5978:	20 20 28 44 29 20 55 6E
\$5980:	6C 6F 63 6B 20 20 28 46
\$5988:	29 20 4F 6E 2D 4C 69 6E
\$5990:	65 20 20 28 48 29 20 50
\$5998:	72 65 66 69 78 20 20 20
\$59A0:	28 4A 29 20 50 72 69 6E
\$59A8:	78 65 72 20 2D 3E 00 11
\$59B0:	11 11 11 11 11 11 11 11
\$59B8:	11 11 11 11 11 11 11 11
\$59C0:	11 11 11 11 11 11 11 11
\$59C8:	11 11 11 11 11 11 11 11
\$59D0:	11 11 11 11 11 11 11 11
\$59D8:	11 11 11 11 11 11 11 11
\$59E0:	11 11 11 11 11 11 11 11
\$59E8:	11 11 11 11 11 11 11 11

## Patch B.15—cont.

```
$59F0: 11 11 11 11 11 11 11 11
$59F8: 11 11 11 11 11 11 11 11
$5A00: 11 11 11 11 11 11 11 11
$5A08: 11 11 11 11 11 11 11 11
```

```
$5A10: 11 11 11 11 (stop at $5A13!)
```

5. [A] Verify: \$591C-5A13 per above.

[B] Verify: \$5A14- 23 51 65

6. BSAVE AWD.SYS, A\$2000, E\$6020, TSYS.

**Patch B.16. The STRETCHIFIER (ProDOS 2.0).**

**For:** ProDOS Applewriter 2.0, version AWD.SYS only. AWD.SYS runs only on the 80 column IIc or the 128K IIfx.

**What it does:** Eliminates "shortlines" caused by counting any imbedded printing commands as real characters.

**How it works:** For every escape character found and actually used in the line, two extra counts are added to the line length. This exactly compensates an imbedded escape command followed by a single character.

**Gotchas:** Uses memory locations from \$59AF-59CF. The SCRUNCHIFIER patch MUST be previously installed. To handle imbedded commands that are longer than an escape and a single character, or non-escape commands, "bank" as many characters as needed. Note that an "[esc][esc]" banks two characters, while an "[esc][null]" banks just one.

**The patch:**

1. Make a third or higher backup copy of ProDOS Applewriter 2.0, using the filer.
2. Get into /BASIC.SYS. CALL -151 to get into monitor.
3. BLOAD AWD.SYS, A\$2000, E\$6020, TSYS,D2
4. [A] Verify: \$49C7- 99 00 1C  
     [B] Change: \$49C7: 20 AF 59
5. [A] Verify: \$49EE- 4C D4 4B  
     [B] Change: \$49EE: 4C BB 59
6. [A] Verify: \$59AF- 11 11 11 11 11 . . .  
     [B] Change: \$59AF: 99  
                   \$59B0: 00 1C C9 1B D0 04 E6 75  
                   \$59B8: E6 75 60 C4 7E F0 0E B9  
                   \$59C0: 00 1C C9 1B D0 04 C6 75  
                   \$59C8: C6 75 88 D0 EE 4C D4 4B
7. [A] Verify: \$49C7-49C9 per above.  
     [B] Verify: \$49EE-49F0 per above.  
     [C] Verify: \$59AF-59CF per above.
8. BSAVE AWD.SYS, A\$2000, E\$6020, TSYS.

**Patch B.17. The PROMPTIFIER (ProDOS 2.0).**

**For:** ProDOS Applewriter 2.0, version AWD.SYS only.  
AWD.SYS runs only on the 80 column IIC or the 128K IIE.

**What it does:** Loads to screen on an unconditional "\" instead of the variable UT. Allows self-prompting glossaries to work all the time.

**How it works:** Compares an immediate "\", rather than to an absolute UT. NOP used to absorb extra byte.

**Gotchas:** Without this patch, self-prompting glossaries will prompt to file, rather than screen unless UT is a "\". The program author must have had a good reason for purposely making the change from "\" to UT on this release. I don't know what that reason is.

**The patch:**

1. Make a third or higher backup copy of ProDOS Applewriter 2.0, using the filer.
2. Get into /BASIC.SYS. CALL -151 to get into monitor.
3. BLOAD AWD.SYS, A\$2000, E\$6020, TSYS,D2
4. [A] Verify: \$396E- CD DE B8  
[B] Change: \$396E: C9 5C EA
5. [A] Verify: \$396E-3970 per above
6. BSAVE AWD.SYS, A\$2000, E\$6020, TSYS.

Patch B.18. *The CURSIFIER (ProDOS 2.0).*

**For:** ProDOS Applewriter 2.0, version AWD.SYS only.  
AWD.SYS runs only on the 80 column IIc or the 128K IIe.

**What it does:** Puts the cursed character into the \$D string.  
Can greatly speed up WPL routines that have to scan characters in sequence. Substitutes a LF for a CR so that carriage returns can be handled as legal \$D characters. Diverts [Q]-H.

**How it works:** Relabels [Q]-H as cursifier. Links [Q]-H with space freed up by SCRUNCHIFIER patch. When activated, reads the cursed character, forces it to low ASCII. If a carriage return, substitutes \$0A or [J] linefeed. Puts character into \$D string and shortens string to one character.

**Gotchas:** [Q]-H does the same thing as the [esc] key, so it may be diverted without any loss. The SCRUNCHIFIER patch MUST be previously installed. Uses memory locations from \$59D0 through \$59E8 freed up by the SCRUNCHIFIER.

**NOTE:** This same technique can be used to link [Q]-H to any custom machine language module of your choosing.

**The patch:**

1. Make a third or higher backup copy of ProDOS Applewriter 2.0, using the filer.
2. Get into /BASIC.SYS
3. BLOAD AWD.SYS, A\$2000, E\$6020, TSYS,D2
4. [A] Verify: \$59D0- 11 11 11 11 . . .  
 [B] Change: \$59D0: 20 75 28 20 12 01 29 7F  
               \$59D8: C9 0D 00 02 A9 0A 8D C0  
               \$59E0: 1E A9 00 8D C1 1E 4C C7  
               \$59E1: 28
5. [A] Verify: \$5DA0: 54 6F 67 67  
 [B] Change: \$5DA0: 43 75 72 73 6F 72 20 2D  
               \$5DA8: 2D 3E 20 24 44 20 20 20  
               \$5DB0: 20 20 20 20 20 20 20 20
6. [A] Verify: \$5E29- 93 35  
 [B] Change: \$5E29: CF 59 (Address MINUS one!)
7. [A] Verify: \$59D0-59E6 per above.  
 [B] Verify: \$5DA0-5DB7 per above.  
 [C] Verify: \$5DB8- 0D  
 [D] Verify: \$5E29-5E2A per above.
7. BSAVE AWD.SYS, A\$2000, E\$6020, TSYS.





---

# C

---

## **Internal ProDOS Applewriter 2.0 Program Details**

## Listing C.1. Detailed script of AW.SYSTEM booting loader.

AW.SYS is automatically booted if it is the first .SYS file on a ProDOS diskette. The program tests the machine, picks a version of Applewriter, installs that version, and then runs the Applewriter code.

There are three possible Applewriter versions:

```
AWB.SYS - 40 columns, 128K
AWC.SYS - 80 columns,  64K
AWD.SYS - 80 columns, 128K
```

This booting code is easily changed to include such things as modem turn-on, parallel card setup, downloading of custom character fonts, etc.

Note that all slot three cards are disconnected by the main program. Thus, prebooting commands to a third-party video or screen card in slot #3 is futile.

The AW.SYS code is installed and run at \$2000 and is a total of 461 bytes long.

.....

\$2000-2007 STEERING CODE

On a cold boot, fall through to version checker code. If a version load fails, jump to the error processor.

\$2008-2037 VERSION CHECKER

Set steering flag to jump to error processor  
 Read the ProDOS machine i.d. byte at \$BF98.  
 If a future machine, use AWD.SYS if 40/80 switch is set to 80. If not, use AWB.SYS.  
 If a present machine, check for machine type.  
 If a II or II+, print wrong version message.  
 If a IIe, check memory size. If 128K, use AWD.SYS. If 64K, use AWC.SYS. If a IIc and 40/80 switch is set to 80, use AWD.SYS. If set to 40, use AWB.SYS. Version is selected by poking B,C, or D into pathname.

Note: for a 40 column, 128K IIe, the 80 column code is used, and screen margins are manually set by the user.

\$2038-204B INSTALL MAIN APPLEWRITER PROGRAM

Set the case flag to upper case only. Open the file named in the pathname. Move an image of the following code to \$0300-0340 so it does not get plowed as the new code loads. Jump to \$0300, reading the program off disk. Then jump to \$2000. If a good load, the newly booted word processing code runs. If not, jump to error processor.

\$204C-205C ProDOS READ MLI

Read the opened file into \$2000. Data file begins at \$0309.

## Listing C.1—cont.

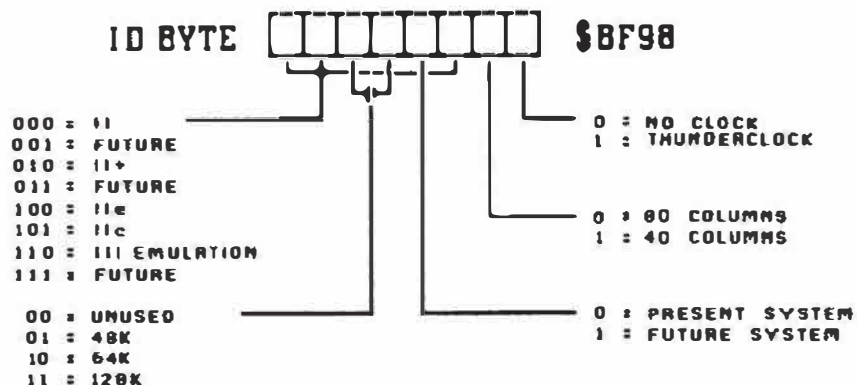
\$205D	STEERING FLAG
	If \$00, check machine and load version. If \$FF, process errors.
\$205E-2072	ProDOS OPEN MLI
	Open the file whose pathname starts at \$201D. Use \$BB00-BEFF as a ProDOS work buffer. Move file number to READ code. Data file starts at \$206E.
\$2075-207D	ProDOS CLOSE MLI
	Close all open files. Data file starts at \$207C.
\$207E	CASE FLAG
	If \$00, print messages in mixed lower and upper case. If \$FF, print upper case only.
\$207F-208C	WRONG MACHINE EXIT
	Trying to boot on a II or II+ gets you here. Close all files. Clear screen. Put down wrong machine message. Jump to error exit.
\$208C-20A4	ERROR PROCESSOR
	Test for no error; if none return to calling code. If an error, clear screen and print error message. Print PRESS RETURN message and wait for user response. Fall through to ProDOS QUIT.
\$20A5-20B2	ProDOS MLI QUIT
	Routine used to transfer control to a new ProDOS SYSTEM program. There is a seven byte buffer needed here and used internally by QUIT.
\$20B3-20D0	MESSAGE PRINTER
	Print the message whose starting address is A (low) and X (high) to screen, in normal text, ending on a \$00 marker. Check case flag. If \$00, print as mixed case. If \$FF, change to all upper case.
\$20D1-20D8	PATHNAME
	Holds the name of the Applewriter file to be booted. Length is seven characters. Can be AWW.SYS, AWC.SYS, or AWD.SYS, depending on poked results of the version checker code.
\$20D9-211A	ERROR MESSAGE
	Low ASCII text stating that an error has occurred, ending on a \$00 marker.
\$211D-21A8	WRONG MACHINE MESSAGE
	Low ASCII text stating that the wrong machine is in use, ending on a \$00 marker.

## Listing C.1—cont.

\$21AB-21CD RETURN PROMPT

Low ASCII text giving a return prompt, ending  
on a \$00 marker.

Here's some extra detail on that ProDOS system i.d. byte at  
\$BF00 . . . .



Additional info on ProDOS appear in Beneath Apple ProDOS  
(Quality Software #61383) or in the ProDOS Technical Manual  
(Apple Computer #A2W0010).

## Listing C.2. ProDOS Appewriter 2.0 low work files.

The low workfiles for ProDOS Appewriter 2.0 sit in main RAM from \$0000 through \$1FFF. These are storage areas that are likely to change as the program is used.

Here are details on the low work files:

**\$0000-00FF -- PAGE ZERO WORK AREA**

Page zero holds pointers, counters, stashes, and flags for program use. This area is so important that we have set aside listing 7.5 and listing 7.6 for full details.

**\$0100-0156 -- MEMORY MANAGEMENT CODE**

Routines to manage auxiliary memory are put here since main pages zero and one are never switched or made inactive in this program.

There are six management routines. These get installed during a cold start and are affirmed on each warm restart:

- \$0100-0108 -- Read screen character from auxiliary memory.
- \$0109-0111 -- Read cursed character from top of LOFILE.
- \$0112-011A -- Read character from bottom of HIFILE.
- \$011B-0123 -- Read character to be printed from LOFILE.
- \$0124-012C -- Read character pointed to by general use pointer.
- \$012D-013D -- Back screen pointer to start of present screen line.

**\$01??-01FF -- 6502 STACK**

The 6502 stack inits to \$01FF and builds down. The stack is used to save the return address on subroutines and separately to temporarily save and restore register values. Since the stack builds down, the area below the stack is risky to use, although values between \$013E and 0180 are probably safe.

**\$0200-02FF -- KEYBUFFER**

Keystrokes are read into this keybuffer, either directly from the keyboard or else from the type-ahead buffer during hectic times, or by the LS load string code module. This is a major work area in which such things as delimiters are processed. Low ASCII characters start at \$0200 and build up in memory, ending with an \$0D carriage return.

**\$0300-037F -- CHARACTER SWALLOW BUFFER**

Single characters being deleted get saved here by the open-apple, left arrow command. They are restored by the open-apple, right



## Listing C.2—cont.

arrow command. The buffer is 128 characters long and works on a round-and-round basis. A pointer at \$AC decides where to put or get the next character. Each ASCII character is put one address higher than the previous one.

\$0380-0385 -- DECIMAL VALUE STASH

Decimal value held here, in left justified ASCII format. Used to input a value for decimal/hex conversion, or to hold the result of a hex/decimal conversion.

\$0386-03CF -- APPARENTLY UNUSED LOCATIONS

\$03D0-03FF -- SYSTEM VECTORS

Addresses of key interrupt, control, soft reset, and reset normally sit here.

Of interest are:

\$03F0 - 03F1 BRK processor address  
 \$03F2 - 03F3 RESET processor address  
 \$03F4 - POWRUP byte (funny EXOR)  
 \$03FE - 03FF IRQ processor address

These are all preset on a cold start so they all vector to the warm restart routine at \$20B4.

\$0400-07FF -- TEXT SCREEN

Characters to appear on the screen are mapped into this memory area. The even characters go in main memory and the odd characters go in auxiliary memory for the 80 column screen.

Note that all screen code is done inside Applewriter 2.0. The usual monitor routines are not used since they are slow, have memory conflicts, do not save keystrokes, cannot do a horizontal scroll, and do not handle screen motions in the way needed.

\$0579 - PORT 1 LINE WIDTH (IIC)  
 \$057A - PORT 2 LINE WIDTH (IIC)  
 \$06F9 - PORT 1 VIDEO ECHO DEFEAT (IIC)  
 \$06FA - PORT 2 VIDEO ECHO DEFEAT (IIC)

\$0779 - PORT 1 LINE WIDTH (future machine)  
 \$077A - PORT 2 LINE WIDTH (future machine)  
 \$07F9 - PORT 2 VIDEO ECHO DEFEAT (future machine)  
 \$07FA - PORT 2 VIDEO ECHO DEFEAT (future machine)

These "screen holes" hold values needed by the built-in ports on the IIC and on a future mystery Apple. The line width is set to \$FF, for 256 characters before forcing a carriage return. The video echo is set to \$00 to defeat video echo during printing. Note that video echo would garble the screen because of custom ProDOS Applewriter 2.0 code.

## Listing C.2—cont.

## \$0800-1000 -- GLOSSARY FILE -or- DISK FORMATTER

All of the glossary entries go into this file area. Low ASCII characters build from the bottom up and each entry ends with a carriage return. Fake carriage returns are stashed with a "]" character if they are needed inside a string. A \$00 marks the end of the last string. Both the cold start and the purge command "empty" this file by putting a \$00 at \$1B00. The glossary is accessed via general use pointer 80,81.

The glossary is completely and destructively overwritten by the FORMAT code used to initialize a new diskette.

## \$1000-1800 -- WPL PROGRAM FILE -or- DISK FORMATTER

A WPL program to be run goes here. Each WPL command consists of a group of low ASCII characters ending with a carriage return. A \$00 value marks the end of the program in case the QT command is missed. The WPL program counter \$A0-A1 reads this file, controlled by the WPL continue flag at \$E7 and the WPL activity flag at \$DF. In use, each WPL statement is read, interpreted, and then carried out. The WPL program file can be 2048 characters long if no footnotes are in use. With footnotes, the file can only be 1024 characters long.

The WPL program file is completely and destructively overwritten by the FORMAT code that is used to initialize a new diskette.

## \$1400-17FF -- FOOTNOTE BUFFER -or- WPL PROGRAM FILE -or- DISK FORMATTER

If footnotes are in use, they are held here from the time the footnote occurs in the text until the bottom of the current page being printed. The ASCII characters build up from \$1400 with each separate footnote ending in a carriage return. A \$00 marks the end of the last footnote. Flag \$FE keeps track of footnote use, with pointer pair \$00,01 locally used to load and then read this file. If footnotes are not in use, then these 1024 locations can be used as additional WPL work file memory.

This file area is completely and destructively overwritten by the FORMAT code used to initialize a new diskette.

## \$0800-17FF DISK FORMATTING CODE

On a [O]-I, a slightly modified ProDOS FORMATTER is loaded into this memory space. The formatting module is then run, starting at \$0800 to format a new diskette. Note that the formatting is for a "file" diskette

## Listing C.2—cont.

and does NOT include the ProDOS operating system.

Note also that any use of the [O]-I format command trashes your glossary, WPL files, and footnote files.

\$1800-1BFF WORD AND PARAGRAPH DELETION BUFFER

Whole words and entire paragraphs are saved and restored to this area with the [W] and [X] commands. A pointer pair at \$94 and \$95 continuously points to the next available location. This is done on a round-and-round basis, with the character after \$1BFF going into \$1800. A separate counter pair of \$EF and \$F0 keeps track of >1024 overflows. A space ends [W], while a carriage return ends an [X] access.

\$1C00-1CFF -- LINE FORMATTING BUFFER

These 256 locations are used to format a line being justified, as well as to hold the searching delimiters during [L]oad, and to handle word wraparound during screen line formatting.

\$1D00-1D3F -- WPL STACK

These 64 locations hold the 32 possible subroutine return addresses for SR commands in WPL. Pointer \$92 accesses these locations a pair at a time, entering a new address pair on each SR and reading a pair on each RT return.

\$1D40-1D7F -- TYPE-AHEAD CHARACTER BUFFER

The 64 locations here hold characters from the time they are typed until the time they can be used, allowing the user to get as many as 64 keys ahead of the program without any errors. A filling pointer \$F3 enters the characters as they are typed. An emptying pointer \$F2 gets the characters as they can be used. During non-hectic times, \$F2 = \$F3 and the buffer is empty. The buffer goes round-and-round, with the next key after \$177F going into \$1740. Should an open-apple or a closed-apple key be used with a normal key, these are separately saved in the apple key buffer at \$1FC0-1FFF.

\$1E00-1E3F -- WPL CHARACTER STRING \$A

\$1E40-1E7F -- WPL CHARACTER STRING \$B

\$1E80-1EBF -- WPL CHARACTER STRING \$C

\$1EC0-1EFF -- WPL CHARACTER STRING \$D

These four work files hold the WPL character strings \$A-\$D. Each string consists of low

## Listing C.2—cont.

ASCII characters building up from the starting address and ending with a \$00 marker. A \$00 at the starting address means this string is not yet in use. All four strings are init'd to \$00 during a cold start. Pointer \$8E acts as a locator to read these strings.

\$1F00-1FFF -- TL,BL FORMATTING BUFFER

Temporarily used in one piece to format the top line and bottom line to their "open" form during printing. Other uses of this buffer area are also temporary, so there is no conflict.

\$1F00-1F3F -- MULTIPLE USE BUFFER

There are two local uses for this buffer. As a pathname hold, it holds the length of the pathname in \$1F00, followed by the path name in low ASCII and ending with a \$00 marker. It is also used to substitute (X) - (Z) WPL numerics into their "open" form.

\$1F40-1F7F -- MULTIPLE USE BUFFER

There are three local uses for this buffer. As a pathname hold, it holds the main or "=" pathname, while other temporary use is made of ProDOS. During RENAME, the old pathname is held here. This same memory space is used as a modem "type ahead" buffer, allowing the modem to not overwrite during such things as screen scrolling at higher baud rates. A filling pointer at \$245D and an emptying pointer at \$245E go round-and-round, similar to the type-ahead character buffer at \$1D40.

\$1F80-1FBF -- FIND STRING SAVE

The "=" search and replace string gets saved here for possible reuse. Using [F] = will repeat the previous search.

\$1FC0-1FFF -- APPLE KEY BUFFER

An auxiliary to the main type-ahead buffer at \$1740-17FF. This area remembers whether an open-apple or a closed-apple key was also pressed at the same time another key was pressed. Flag \$FB holds this status on reading the type-ahead buffer pair.



Listing C.3. *ProDOS Applewriter 2.0 high work files.*

The high workfiles for ProDOS Applewriter 2.0 sit in main RAM from \$B600 through \$BFFF. These also are storage and buffer areas that are likely to change as the program is used.

Here are details on the high work files:

**\$B600-B6FF TAB STATUS IMAGE**

The image of the tab status line is held here. Unset tabs are held as high ASCII characters which appear on-screen as normal text. Set tabs are held as low ASCII characters which appear on-screen as inverse text. The length of the "fives" tab marker gets longer at 100, and longer still at 200.

**\$B700-B77F PATHNAME HOLD**

The ProDOS pathname currently being used is held here. The old, or "main" pathname is saved to the "=" pathname save at \$1FC0 during glossary, WPL, TAB or PRT disk access.

**\$B780-B7BF APPARENTLY UNUSED LOCATIONS**

**\$B7C0-B8FF PRT FILE**

A ".PRT" file, as saved or read from disk gets held here. There are three parts of this file, the top line, the bottom line, and the print constants, detailed below.

**\$B7C0-B83F TOP LINE**

The top line is held here in its "compact" form, with delimiters and "#" as a page number.

**\$B840-B8BF BOTTOM LINE**

The bottom line is held here in its "compact" form, with delimiters and "#" as a page number.

**\$B8C0-B8FF PRINT AND MODEM CONSTANTS**

The print and modem constants are held here, as set by the operator or as read from a .PRT file. Most constants are assumed to have a range of 0 to 65535. One exception is the PM or paragraph margin, whose negative values are held as 2's complement signed binary.

These print and modem constants are stashed in hex in the usual 6502 "low-high" format. Thus a print constant of decimal 255 or less will use only the first byte of each pair; while the second byte will remain at \$00.

Here are the specific location pairs:

\$B8C0 - LM Left Margin  
 \$B8C2 - PM Paragraph Margin  
 \$B8C4 - RM Right Margin  
 \$B8C6 - TM Top Margin  
 \$B8C8 - BM Bottom Margin  
 \$B8CA - PN Page Number

## Listing C.3—cont.

\$B8CC - PL Printed Lines  
 \$B8CE - PI Page Interval  
 \$B8D0 - LI Line Interval  
 \$B8D2 - SP Space Interval  
 \$B8D4 - PD Print Destination  
  
 \$B8D6 - WPL (x) numeric value  
 \$B8D8 - WPL (y) numeric value  
 \$B8DA - WPL (z) numeric value

\$B8DC - Modem Carriage Return delay  
 \$B8DE - Underline token (ASCII)

\$B8E0 - Justify flag:  
     00 - Fill Justify  
     01 - Left Justify  
     02 - Right Justify  
     03 - Center Justify

\$B8E2 - Serial Slot 1 Baud Rate, etc.  
 \$B8E3 - Serial Slot 1 Parity Set  
 \$B8E4 - Serial Slot 2 Baud Rate, etc.  
 \$B8E5 - Serial Slot 2 Parity Set

**\$B8E6-B900 APPARENTLY UNDEFINED PRINT VALUES**

These apparently unused locations get loaded and saved as if they were print values. Start at the high end of you divert them.

**\$B900-BAFF SECTOR IMAGE FOR LOAD/STORE**

A 512 byte image of a ProDOS disk sector goes here so it can be scanned for starting and ending delimiters during a load or store. This area is separately used as a pathname hold.

**\$BB00-BDFF ProDOS WORK BUFFER**

A 1K work buffer, used internally by the ProDOS operating system, sits here. This buffer area is first set aside by the AW.SYSTEM boot code.

**\$BE00-BEFF ProDOS MLI SYSTEM PAGE**

Usually used by ProDOS for BASIC interfacing. While not used by Applewriter, this page is preserved so that other .SYS programs can hold values here during Applewriter activity.

**\$BF00-BFFF ProDOS MLI GLOBAL PAGE**

All access to ProDOS is done by linking through this page. Important values include:

\$BF00 - A subroutine call here links to various ProDOS routines, by reading a data block of parameters that follow the subroutine call.

\$BF16 - Slot 3, D1 driver address  
 \$BF26 - Slot 3, D2 driver address  
 \$BF30 - Slot and drive of last used device  
 \$BF31 - Count of active devices  
 \$BF32 - Start of active device list  
 \$BF98 - Machine ID byte (see Table 7.1)



## Listing C.4. ProDOS Applewriter 2.0 internal files.

The internal files of ProDOS Applewriter 2.0 are two main types. Some are local value stashes used as auxiliary working registers. Others are tables of parameters used to access various ProDOS interface MLI routines.

The internal files are important, first because they are essential to understanding ProDOS access. Secondly, if you attempt to capture your own source code, these file areas must be carefully bypassed, or else aliasing, "starting on the wrong foot", or illegal op-codes will result.

Here are details on the internal files:

**\$20DE-20DF LOW MEMORY BOUNDARY**

The beginning of the LOFILE area is held here. This is a read only location, set to auxiliary memory \$0800 in this version. Low-high.

**\$214F-2151 SLOT 3 HOLD**

These three bytes hold the address of any slot 3 drivers during Applewriter access. This allows another SYS program to transfer to and use Applewriter, and then return, keeping its own slot 3 setup intact.

**\$23A9- GET LINE SOURCE FLAG**

If \$00, gets line from user via type-ahead buffer \$1D40. If \$FF gets line from modem via receive-ahead buffer \$1F40.

**\$254D- RECEIVE-AHEAD BUFFER FILLER  
\$254E- RECEIVE-AHEAD BUFFER EMPTIER**

These two pointers control the modem's receive-ahead buffer, allowing characters to be input without loss during screen scrolls. If the two pointers are equal, the character is directly used. If not, \$245D fills and \$245E empties. These pointers are restricted to a 64 character range of \$00-3F and are added to buffer base address \$1740.

**\$2590-25BF BASH TABLE**

The leftmost base address of each vertical screen line is held in this table. Line zero has a base address of \$0400, line one \$0480, and line twenty-three \$07D0. Looking up base addresses is much faster than calculation.

**\$2626- LOCAL Y SAVE**

Local and temporary stash of Y register during character-to-file entry.

## Listing C.4—cont.

\$2AEC-	GLOSSARY NEST POINTER
	An eight-level pointer, \$00, \$02, \$04 ... that points to an address in the glossary nest below.
\$2AED-2AFC	GLOSSARY NEST STACK
	Holds up to eight pointers to the glossary at \$0800. Used to let one glossary entry call another. The pointer at \$2AEC remembers which level of glossary access is currently active.
\$2C86-2C88	ProDOS MLI GET PREFIX LINK
	Used by catalog. Command \$C7. Buffer \$2C8C
\$2C8C-2C8E	ProDOS MLI GET PREFIX FILE
	One parameter; use pathname at \$1F00.
\$2D08-2D0A	ProDOS MLI QUIT LINK
	Used by quit. Command \$65. Buffer \$2D0C.
\$2D0C-2D12	ProDOS MLI QUIT FILE
	Four parameters; six reserved \$00 locations.
\$2E70-2E73	MODEM ACTIVITY STASH
	Four values held here control modem actions:
	\$2E70 - \$00=none \$40=active \$80=slave
	\$2E71 - RESQ in-process flag.
	\$2E72 - RESQ ASCII stash
	\$2E73 - {R}ecord flag
\$2EB2-2EB4	ProDOS MLI WRITE LINK
	Used by glossary. Command \$CB. Buffer \$2EBF.
\$2EBF-2EC6	ProDOS MLI WRITE FILE
	Four parameters; reference #\$01; data buffer \$0800; request length poked by \$2E95; actual length poked by ProDOS.
\$2F73-2F75	ProDOS MLI WRITE LINK
	Used to save tabs. Command \$CB. Buffer \$2F7C.
\$2F7C-2F83	ProDOS MLI WRITE FILE
	Four parameters; reference #\$01; data buffer \$1D80; request length \$0080; actual length poked by ProDOS.
\$2F9D-2F9F	ProDOS MLI READ LINK
	Used to load tabs. Command \$CA. Buffer \$2FA9.

## Listing C.4—cont.

\$2FA9-2FB0	<p>ProDOS MLI READ FILE</p> <p>Four parameters; reference #\$01; data buffer \$1D80; request length \$0080; actual length poked by ProDOS.</p>
\$2FCA-2FCC	<p>ProDOS MLI WRITE LINK</p> <p>Used to save PRT. Command \$CB. Buffer \$2FD3.</p>
\$2FD3-2FDA	<p>ProDOS MLI WRITE FILE</p> <p>Four parameters; reference #\$01; data buffer \$B780; request length \$0170; actual length poked by ProDOS.</p>
\$2FF4-2FF6	<p>ProDOS MLI READ LINK</p> <p>Used to load PRT. Command \$CA. Buffer \$2FFD.</p>
\$2FFD-3003	<p>ProDOS MLI READ FILE</p> <p>Four parameters; reference #\$01; data buffer \$B780; request length \$0170; actual length poked by ProDOS.</p>
\$3048-304A	<p>ProDOS MLI READ LINK</p> <p>Used to load formatter. Command \$CA. Buffer \$305B. Note: destructive overwrite of glossary, WPL, and footnotes.</p>
\$305B-3062	<p>ProDOS MLI READ FILE</p> <p>Four parameters; reference #\$01; data buffer \$0800; request length \$1200; actual length poked by ProDOS.</p>
\$30EB-30ED	<p>ProDOS MLI OPEN LINK</p> <p>Used by file setup. Command \$C8. Buffer \$30EF.</p>
\$30EF-30F4	<p>ProDOS MLI OPEN FILE</p> <p>Three parameters; pathname at \$1F00; data buffer at \$BB00; reference number.</p>
\$30F5-	<p>REFERENCE NUMBER STASH</p> <p>Internal Applewriter hold of reference number of last opened file.</p>
\$30FC-30FF	<p>ProDOS CREATE MLI LINK</p> <p>Used to create directory. Command \$C0. Buffer \$3102.</p>
\$3102-310D	<p>ProDOS CREATE MLI FILE</p> <p>Seven parameters; pathname \$1F00; may be destroyed, renamed, written, or read; file type is a directory; \$00 auxiliary type; linked subdirectory possible.</p>

## Listing C.4—cont.

\$3114-3116	ProDOS CREATE MLI LINK
	Used to create text file. Command \$C0. Buffer \$311A.
\$311A-3125	ProDOS CREATE MLI FILE
	Seven parameters; pathname \$1F00; may be destroyed, renamed, written, or read; file type is a textfile; \$00 auxiliary type; standard seedling possible.
\$3252-3254	ProDOS MLI READ LINK
	Used to read one sector of a text file. Command \$CA. Buffer \$325D.
\$325D-3264	ProDOS MLI READ FILE
	Four parameters; reference #\$01; data buffer \$B900; request length \$0200; actual length poked by ProDOS.
\$3647-	ADJUST FORMAT FLAG
	Use {S}ave code as {S}ave if \$00. Reformat screen margins only if set to \$FF.
\$37CD-37CF	ProDOS MLI WRITE LINK
	Used to save textfiles, one segment at a time. Command \$CB. Buffer \$37D3.
\$37D3-37DA	ProDOS MLI WRITE FILE
	Four parameters; reference #\$01; data buffer \$B900; request length \$0200; actual length poked by ProDOS.
\$37DB-37DC	FILE POSITION HOLD
	Holds the position in the textfile at which another sector is to be transferred to disk.
\$37EE-37F0	ProDOS CLOSE MLI LINK
	Used to close all files. Command \$CC. Buffer \$37F7.
\$37F7-37F8	ProDOS CLOSE MLI FILE
	One parameter. Close all files on \$00 reference number.
\$37FC-37FE	ProDOS MLI GET EOF LINK
	Used by append. Command D1. Buffer \$380C.
\$380C-3811	ProDOS GET EOF MLI FILE
	Two parameters. File reference number, followed by three byte result. Third byte usually \$00.
\$3826-3828	ProDOS SET EOF MLI LINK
	Used by Append. Command \$CE. Buffer \$382C.

## Listing C.4—cont.

\$382C-3830	ProDOS SET EOF MLI FILE
	Two parameters. File reference number, followed by three byte ending address. Third byte is usually \$00.
\$38D3-38D5	PATHNAME PARAMETERS
	Holds parameters used by pathname routines:
	\$38D3 - Slot and drive as DSSS 0000
	\$38D4 - Filename length to "/" delimiter
	\$38D5 - Slot number *16
\$38DC-38DE	ProDOS MLI ON-LINE LINK
	Used by on-line routine. Command \$C5. Buffer \$38E2.
\$38E2-38E5	ProDOS MLI ON LINE FILE
	Two parameters. Slot and drive as DSSS 0000; Result to buffer \$B900.
\$38F9-38FB	ProDOS MLI OPEN LINK
	Used by file setup. Command \$C8. Buffer \$30EF. Buffer shared with earlier OPEN routine.
\$391E-3920	ProDOS SET EOF MLI LINK
	Used during setup Command \$CE. Buffer \$3924.
\$3924-3928	ProDOS SET EOF MLI FILE
	Two parameters. File reference number, followed by three byte ending address. Third byte is usually \$00.
\$3C4F- \$3C50-	LOCAL Y SAVE LOCAL X SAVE
	Used as temporary register stash by [F]ind.
\$439D-439F	ProDOS MLI WRITE LINK
	Used to write one byte to a disk text file at a time under PD8 print to disk. Command \$CB. Buffer \$43A3.
\$43A3-43AA	ProDOS MLI WRITE FILE
	Four parameters; reference number; data buffer \$43AB; request length \$0001; actual length poked by ProDOS.
\$43AB-	PRINT TO DISK DATA BUFFER
	Holds single character being written to disk under PD8 print-to-disk option.



## Listing C.4—cont.

\$4CA4-4CA7	TIME DELAY TABLE	
		Holds time delay values needed for low modem baud rates. Delay values approximately equal to 7, 10, 20, 40, and 80 milliseconds. When combined with overhead, these translate to maximum baud rates under 2400, 1200, 600, 300, and 150 baud respectively.
\$3C4F- \$3C50-	LOCAL Y SAVE LOCAL X SAVE	
		Used as temporary register stash by screen printing routine.
\$4EF5-4F14	BAUD RATE LIST	
		Hexadecimal list of baud rates 0, 50, 75, 135, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 9600, and 19,200. In usual 6502 low-high order.
\$504C-	ON-LINE STASH	
		Numeric stash holds number of volumes on line.
\$509F-50A1	ProDOS MLI RENAME LINK	
		Used by rename. Command \$C2. Buffer \$50A5.
\$50A5-50A9	ProDOS MLI RENAME FILE	
		Two parameters. Old filename at \$1F40. New filename at \$1F00.
\$50CB-50CD	ProDOS MLI GET ATTRIBUTES LINK	
		Used by lock and unlock. Command \$C4. Buffer at \$50E2, and shared with the Set Attributes ProDOS link.
\$50DC-50DE	ProDOS MLI SET ATTRIBUTES LINK	
		Used by lock and unlock. Command \$C3. Buffer at \$50E2, and shared with the Get Attributes ProDOS link.
\$50E2-50FC	ProDOS ATTRIBUTES FILE	
		Seven parameters. Filename at \$1F00; Access \$C3 can write, read, destroy, or rename; \$01 can read only (locked); File type; Aux file type; Null field; Modified date; Modified time.
\$5106-5108	ProDOS DESTROY MLI LINK	
		Used by delete. Command \$C1. Buffer \$510C.
\$510C-510E	ProDOS DESTROY MLI FILE	
		One Parameter. Pathname at \$1F00.
\$5117-5119	ProDOS SET PREFIX LINK	
		Used to set prefix. Command \$C6. Buffer \$5120.



## Listing C.4--cont.

\$5120-5122	ProDOS SET PREFIX FILE
	One parameter. Prefix name at \$1F00.
\$5123-	CATALOG TO MEMORY FLAG
	If \$00, do catalog to screen. If \$FF, do a catalog to text file.
\$5157-515B	ATTRIBUTE BUFFER
	Used during catalog calculations:
	\$5157-5158 Auxiliary Data
	\$5159- Type of File
	\$515A-515B Blocks in Use
\$5318-	LOCAL X6 STASH
	Used to multiply a pointer by six for ProDOS file type access. Pointer*2 is held here and then added to Pointer*4 to get Pointer*6.
\$5398-	LOCAL TIME FLAG
	If \$00, catalog is handling year and month.
	If \$02, catalog is handling hours and minutes.
\$53AA-	PATHNAME LENGTH STASH
	Used locally in printing file names or volume names to the screen.

## Listing C.5. AWD.SYS reference files.

The reference file area holds values that are seldom, if ever, changed. This file area sits above the main program from \$5475 through \$5FFF. See listing 7.7 for specific address location vectors.

Here are details:

\$5475-554D -- FUNCTION LIST

An ASCII list of all available control commands, along with any bottom line prompts needed for those commands. Note that [X] is the delete key and that [M] is a carriage return. [H] is a backspace and [U] is a frontspace. [I] is an actual tab. [K] is a linefeed up, while [J] is a line feed down. This table is scanned in sequence for a match character between the "[" and "]". If a match is found, a bottom line prompt is included on certain commands.

\$554E-5582 -- FUNCTION ADDRESS LIST

Holds the module addresses MINUS ONE for the control commands [X] through [Z]. Thus, the [X] or delete module starts at \$2AD0, [A] for adjust margins starts at \$4F70, and so on.

\$5583-55AB -- PRINT CONSTANTS MATCH FILE

A file of pairs of ASCII characters from LM, PM, ... through RJ, and finally CJ. On a match to a character pair, the entered value is saved to the proper PP slot.

\$55AC-55CE -- WPL COMMAND MATCH FILE

A file of pairs of ASCII characters from GO, DO, ... through LS, and finally EP. On a match to a character pair, a jump is done to the module that handles that WPL command.

\$4C92-4CB5 -- WPL COMMAND ADDRESS FILE

These address pairs hold the starting point MINUS ONE of the WPL modules DO through EP. For instance, the GO module starts at \$465D, DO begins at \$44DC, and EP starts at \$480E.

\$55F4-5732 -- ASCII PROMPTS

These ASCII messages range from "Insert sheet, press return" through "Delete old \$ (Yes/No)?" Proceed / Y They are selected as needed to prompt or inform the user.

\$5733-5858 -- ProDOS ERROR MESSAGE FILE

A list of all ProDOS error messages from #26 "Invalid Operation" through #57 "Duplicate Volume". Text is high ASCII, while error numbers are low ASCII.

## Listing C.5—cont.

## \$5859-5889 -- MORE ASCII PROMPTS

Two additional messages are here, one to announce a ProDOS error, the second the prompt used by {F}ind.

## \$588A-5916 -- WPL ERROR MESSAGE FILE

This stash holds all the WPL error messages, starting with a prompt of "WPL Error:" and ending with "Glossary nesting". When an error happens, the prompt is put down, and is then followed by one of the error messages.

## \$5917-591A -- TAB SCP STASH

Three characters kept here for tab set, clear, or purge command matching.

## \$591B-5A13 -- ProDOS FUNCTIONS MENU

This stash holds all the ASCII characters needed for the [O] DOS access command menu.

## \$554E-5582 -- ProDOS COMMAND ADDRESS LIST

Holds the module addresses MINUS ONE for the ProDOS commands [O]-A through [O]-J. Thus, the [O]-A catalog routine starts at \$5124, etc. through the address list.

## \$5A2A-5A80 -- CONFIGURATION ASCII PROMPTS

Messages used by [O]-J, prompting for slot set and format changes.

## \$5A83-5A88 -- PARITY KEY STASH

Eight match characters for Space, Mark, Even, Odd, or None. None is repeated four times to simplify scanning code.

## \$5A8B-5B59 -- STILL MORE ASCII PROMPTS

ASCII messages to adjust display margins, exit prompt, busy adjusting prompt, ProDOS prompt, blocks report, and return prompt.

## \$5B5A-5BDD -- ProDOS FILE TYPES LIST

A list of sixteen ProDOS file types, ranging from #0 Unknown to #F System. Used by catalog routines.

## \$58DE-5C2B -- CATALOG SCREEN HEADER

An image of the top line of the screen catalog.

## \$5C25-5C2B -- HELP PATHNAME

Holds a HELP 80 image used to get the main help menu off disk.

## Listing C.5—cont.

## \$5C2C-5C2E -- MATCH CHARACTERS

Holds Y for yes, N for no, A for all match characters needed to evaluate user commands.

## \$5C43-5CC8 -- STARTUP SCREEN

Contains an ASCII image of the first screen displayed on a cold start. The three high ASCII imbedded values are the horizontal tab moves used to format the title box.

## \$5CC9-5E1A -- ADDITIONAL FUNCTIONS MENU

Contains an ASCII image of the additional functions menu and selection prompt.

## \$5E1B-5E2C -- ADDITIONAL FUNCTION ADDRESSES

Holds the address MINUS ONE needed to enter each additional function routine, ranging from \$2F84 to load the tab file through \$2C97 to quit.

## \$5E2F-5E32 -- EXPRESS CURSOR MOTION WORKFILE

This misplaced workfile holds two stashes used by the express cursor motions. \$5210 is a line counter, usually set to 12 lines. \$5211 is an abort file, holding an \$A0 for stop on space, or \$00 for stop on file end.

## \$5E33-5E34 -- PRINTING WORKFILE

This misplaced workfile holds two stashes used by the [P]rint routines. \$5212 holds a copy of the left margin LM value, while \$5213 keeps the right margin RM value. These two are subtracted to find the default line length for the top and bottom margins.

## \$5E33-5FEF -- PRINT/PROGRAM FUNCTIONS MENU

Contains the ASCII image of the print/program functions screen. Values are added to this background display during the PP "?" command.

## \$5FF0-5FFF -- APPARENTLY UNUSED LOCATIONS.

Original program load ends at \$5FFF. Patches shown in the previous module extend code as far as \$6020.

Listing C.6. Summary of AWD.SYS page zero use.

A summary of page zero use follows, with much more detail on each location provided in Listing 7.6.

Locations marked with an "\*" have more than one use and must be approached with caution.

Note that location \$24 is neither used nor accessed by ProDOS Applewriter 2.0. This can cause problems with certain "intelligent" printer cards. See Module six for more on this.

-- counters --

\$06 - Top line and bottom line counter  
 \$7C - Vertical line counter  
 \$8A - Screen horizontal position counter  
 \$96,97 - Tab counter  
 \$A0-A1 - WPL program counter  
  
 \$BC - Tab over counter  
 \$BD - Memory page counter  
 \$BE,BF - Running page counter  
 \$EF,F0 - [W] [X] overflow counter  
  
 \$FE - Footnote line counter

-- flags --

\$32 - Inverse flag  
 \$71 - Main/Auxiliary flag  
 \$72 - Verbatium flag  
 \$74 - Carriage return display flag  
 \$76 - End of Print file flag  
  
 \$77 - Copy from memory flag  
 \$78 - Page/Position flag  
 \$79 - String source flag  
 \$7A - Reprompt flag  
 \* \$7F - Multi-use local flag  
  
 \$A8 - Arithmetic mode flag  
 \$AD - String source flag  
 \$B0 - Filename source flag  
 \$B4 - Startup flag  
 \$B8 - Printer enable flag  
  
 \$B9 - HIFILE/LOFILE flag  
 \$C4 - Case flag  
 \$C9 - Any length flag  
 \$CD - Reformatting flag  
 \$CE - Screen source flag  
  
 \$CF - Data direction flag  
 \$DF - WPL/Glossary flag  
 \$E0 - Underline flag  
 \$E1 - Wraparound flag  
 \$E2 - Case flag  
  
 \$E5 - Data line toggle flag  
 \$E7 - WPL continue flag (?)  
 \$E8 - Case change flag  
 \$ED - Mystery flag  
 \$EE - [W] [X] overflow flag

## Listing C.6—cont.

```

$F5 - Replace flag
$F6 - String $A-$D flag
$F7 - Screen display flag
$F8 - Split screen flag

$FD - Screen load flag
$FF - Bottom of page flag

-- pointers --

* $00,01 - Multi-use local pointer
$05 - WPL label pointer
$26,27 - Screen vertical scrolling pointer
$28,29 - Screen base address BASH pointer
$36,37 - COUT destination pointer

$38,39 - KEYIN pointer (never used)
$42,43 - ProDOS BRK default pointer
$44,45 - ProDOS buffer pointer
* $80,81 - Main utility pointer
$84,85 - LOCURS pointer

$86,87 - HICURS pointer
$88,89 - Screen textfile pointer
$8E - String $A-$D pointer
$90,91 - Printer textfile pointer
$92 - WPL subroutine stack pointer

$94,95 - [W] [X] deletion pointer
$98,99 - Static cursor pointer
$9E,9F - Print destination pointer
$A4 - End delimiter pointer
$AA,AB - Prompt pointer

$AC - Swallow buffer pointer
* $AE,AF - Auxiliary utility pointer
$BA,BB - Reformatting pointer
$F2 - Type-ahead emptying pointer
$F3 - Type-ahead filling pointer

* $F9,FA - Deletion or split screen pointer

-- stashes --

* $00 - Multi-use local stash
* $02 - Multi-use local stash
* $03 - Multi-use local stash
$20 - WNDLFT left window margin
$22 - WNDTOP top window margin

$23 - WNDBOT bottom window margin
* $34 - Global Y register save
* $35 - Global X register save
* $75 - Multi use local stash
$7D - Last printable line

* $7E - Multi-use local stash
* $80 - Multi-use local stash
* $81 - Multi-use local stash
* $82 - Multi-use local stash
* $83 - Multi-use local stash

$8B - Bash needed stash

```



## Listing C.6—cont.

\$8C	-	WPL current character stash
\$9A, 9B	-	Memory left stash
* \$A2	-	Multi-use local stash
\$A3	-	Multi-use local stash
\$A6	-	Vertical screen position
\$A7	-	Horizontal screen position
\$B1	-	Screen right margin
\$B2	-	Screen left margin
\$B5	-	Cursor symbol stash
\$B7	-	Space left on line stash
* \$C0	-	Multi-use local stash
\$C1	-	Hexadecimal intermediate byte
* \$C2	-	Multi-use local stash
* \$C0-C2	-	Hexadecimal stash
* \$C5	-	Local Y register save
* \$C6	-	Local X register save
* \$C7	-	Local accumulator save
\$C8	-	Screen centering stash
\$CA	-	Horizontal scrolling trigger
\$CB	-	Right screen margin
\$DE	-	Slot number stash
\$E6	-	Delimiter character stash
\$E9	-	Filename length count
\$EA	-	Wildcard special delimiter
\$EB	-	Carriage return special delimiter
\$EC	-	Any length special delimiter
\$F4	-	Busy prompt
\$FB	-	Apple key stash
\$FC	-	Machine i.d. stash

## Listing C.7. Detailed script of AWD.SYS page zero use.

Page zero is used for pointers that hold address values, counters that keep track of positions, stashes that hold constants or characters, and flags that remember conditions or modes. Single page zero locations are used for eight or fewer bits of information. Double page zero location pairs are used to hold nine to sixteen bits of information.

On a cold boot, all page zero locations above \$60 are set to \$00 values. A \$00 flag value usually means "don't".

Here is a rundown . . .

\$00 -- MULTI USE LOCAL STASH

Local stash when used by itself. On cold boot, a pointer to clear page zero. Error number hold for error processor. Delimiter hold for [S]ave and [L]oad. Working variable in direction check when moving characters. Horizontal tab value for status line. Numeric stash during relative value calculations. Delimiter hold for string comparisons. String identifier for [p]in and length limit for [P]as.

\$01 -- MULTI-USE LOCAL STASH

Local stash when used by itself. A width counter when drawing first screen boxes. Other versions use this as a 40 column offset calculator.

\$00,01 -- MULTI-USE LOCAL POINTER

A 16-bit wide pointer when used as a pair. Used on cold start to fill glossary with carriage returns. Used as glossary pointer on [G]?. Pointer to auxiliary functions. Running hold of lowest possible tab value. With \$02, a filename pointer under [O]a.

\$02 -- MULTI-USE LOCAL STASH

End character hold for [W] and [X]. Volume and filename steering prompt. Tab possible stash. Local stash on [F]ind reprompting. \$A-D string offset and pointer. With \$03, a pointer to the error message file.

\$03 -- MULTI-USE LOCAL STASH

String substitution flag (\$00 = ok) under [P]. High byte of error message file pointer. Tab possible flag if \$FF. No string needed flag with WPL error processor. Valid string stash for [P]in. Single page string flag.

\$05 -- WPL LABEL POINTER

Used as pointer to find a WPL label in [P]go and for LABEL NOT FOUND error message.

## Listing C.7—cont.

```

$06  -- TL/BL CHARACTER COUNTER

      Used when formatting the left, center, and right
      portions of the top and bottom lines.

$20  -- LEFT WINDOW MARGIN

      Inits to $00.  Never changes.  Full screen width
      used.

$22  -- TOP WINDOW MARGIN

      Inits to $00.  Resets to $00 on screen prompt.
      Set to $01 for [Q]I to make room for modem
      prompts.  Calculated value used for split screen
      and status line options.

$23  -- BOTTOM WINDOW MARGIN

      Inits to $18, equal to full 24 screen lines.
      High split screen sets to 12 screen lines.

$24  -- NOT USED (!)

      Some parallel cards will expect to find a value
      here that equals the horizontal cursor position
      on screen.  NO USE OF THIS LOCATION IS MADE BY
      AWD.SYS!  This causes the "problem" cards to
      insert random strings of spaces or ignore most
      imbedded commands.  The cure to these hassles is
      card dependent.

$26,27 -- SCREEN VERTICAL SCROLL POINTER

      Holds the destination address during vertical
      screen scrolling.  Inits to present line position.
      Line below is mapped up one, repeating as needed
      to complete scrolling.

$28,29 -- SCREEN BASE ADDRESS POINTER

      Holds the memory address of the leftmost
      position on the current screen line.  Found by
      table lookup from the BASH table at $2590-25BF.
      Used to enter characters, read from screen, flash
      cursor, and in screen scrolling.

      IMPORTANT NOTE: All screen routines are done
      internally by AWD.SYS.  No use is made of any
      monitor routines.

$32  -- INVERSE FLAG

      An $FF here puts normal text on screen.  A $7F
      puts inverse text on screen.  Used to print
      spaces as white boxes during first screen.

$34  -- GLOBAL Y SAVE STASH

      Used to save the contents of the Y register in
      high level routines.

$35  -- GLOBAL X SAVE STASH

      Used to save the contents of the X register in
      high level routines.

```

## Listing C.7—cont.

## \$36,37 -- COUT DESTINATION POINTER

Initials to "brick wall" RTS. Sets to \$C100, \$C200 etc. for printer or modem. Interface modifies for exact address. Sets to \$4397 for print to disk or \$4415 for print to screen. Note that no use is made of monitor routines \$FDED or \$FDF0 by this program.

## \$38,39 -- KEYIN POINTER

Initials to a "brick wall" RTS. Stays that way throughout program. Monitor KSWL,H routines are not used by this program.

## \$42,43 -- ProDOS DEFAULT BRK POINTER

Reset to \$03F0 upon exit of program. In unusual high-low form.

## \$44,45 -- ProDOS DEFAULT BUFFER POINTER

Reset to \$B900 upon exit of program. In standard low-high form.

## \$71 -- MAIN/AUXILIARY FLAG

If \$00, load into or save from main memory for TAB, PRT, glossary, WPL, or textfile loads and saves involving delimiters. If \$FF load into or save from auxiliary memory for non-delimited text material or for [P]ls.

## \$72 -- VERBATUM FLAG

If \$FF, insert control commands directly in textfile. If \$00, process control commands in the normal way.

## \$74 -- RETURN DISPLAY FLAG

If \$00 do not display carriage returns. If \$FF, display carriage returns as inverse "M" characters.

## \$75 -- MULTI-USE STASH

End of word marker for [F]ind when using all occurrence. Line width for TL and BL. RM-LM for justify routines.

## \$76 -- END OF PRINT FILE FLAG

If \$00, more characters remain in the text file during printing. If \$FF, the text file has all been printed.

## \$77 -- COPY FROM MEMORY FLAG

If \$00 read from disk. If \$00, copy from memory.

## \$78 -- PAGE/POSITION FLAG

If \$FF, display page/position on screen bottom. If \$00, do not display page/position.

## Listing C.7—cont.

```

$79  -- STRING SOURCE FLAG

      If $00, get new string from keyboard.  If $FF,
      use existing string in $0200 keybuffer.

$7A  -- REPROMPT FLAG

      If $FF, reprompt screen bottom for [F]ind.  If
      $00, do not reprompt.

$7C  -- VERTICAL LINE COUNTER

      Keeps track of lines already printed.  Advances
      one count on each printed line.

$7D  -- LAST PRINTED LINE NUMBER

      Used to set lower page limit of body printing.
      Includes body but not footnotes.  Decrementd
      twice for the first footnote line, and once for
      each additional footnote line.

$7E  -- MULTI-USE LOCAL STASH

      Holds the first character offset for center or
      right justification.  Seperately holds the line
      length already printed during wraparound check.

$7F  -- MULTI-USE LOCAL FLAG

      First line in paragraph flag if $FF; otherwise
      $00.  Don't fill justify last line in paragraph
      line if $FF; otherwise $00.  Stop underlining
      flag for UT if $FF; otherwise $00.  Suprisingly,
      these uses do not conflict with each other.
      Setting the flag on the last paragraph line both
      cancels justification and sets up the PM use on
      the next line.  As the same time, underlining is
      not allowed to carry over from paragraph to
      paragraph.  Neat.

$80  MULTI-USE LOCAL STASH

      When used by itself, a local slot number * 2 in
      [O]-J printer setting.  A local Y register save
      during [Q]-B rename.  A local stash of the number
      of digits to print under [O]-A catalog.

$81  -- MULTI-USE LOCAL STASH

      When used by itself, holds the compressed modem
      port data during formatting.  Seperately used
      locally to hold a space character for comparision
      during [O]-A catalog.

$80-81 -- MAIN UTILITY POINTER

      When not seperately used as local stashes.  The
      glossary access pointer, starting at $0800.
      The line totalizer for the page/position display.
      Pointer to control prompts such as [F]ind, etc.
      Transferred and held by $AA-AB.  The [N]ew
      prompting pointer.  Background pointer for
      [P]rint/Program menu.

```



## Listing C.7—cont.

```

$82      -- MULTI-USE LOCAL STASH

        Used with $83 as copy from memory pointer pair.
        Holds modem parametters PPP- ---- during [O]-J
        formatting and setup. Volume name pointer in
        [O]-F On-line. Screen line counter for [O]-A
        catalog. Puts down 15 catalog lines for full
        screen or 7 catalog lines for split screen.

$83      -- MULTI-USE LOCAL STASH

        Used with $82 as copy from memory pointer pair.
        Holds modem parameters SDD1 BBBB during [O]-J
        formatting and setup. File type hold for [O]-A
        catalog.

$84-85   -- LOCURS TEXTFILE POINTER

        Points to the high and open end of the LOFILE
        text area. Used for every major access to LOFILE
        all textfile entries, and all cursor motions.

$86-87   -- HICURS TEXTFILE POINTER

        Points to the low and open end of the HIFILE text
        Used for insertions, deletions, moves, and any
        other time the cursor is not at the end of the
        text file.

$88-89   -- SCREEN TEXTFILE POINTER

        Points to the location in the textfile about to
        be put on the screen. Inits to six or twelve
        lines shy of LOCURS, depending on screen split.
        Keys on high ASCII markers at the end of each
        screen line in the LOCURS and HICURS textfiles.
        Maps LOFILE up to center of screen and cursed
        location. Then switches to HIFILE and maps the
        rest of the screen.

$8A      -- SCREEN HORIZONTAL POSITION COUNTER.

        Counts characters on line during screen update.
        Used to switch from LOFILE to HIFILE at cursed
        position.

$8B      -- BASH NEEDED STASH

        Compares against vertical position A6. If
        different, calculates a new screen base address
        via the BASH routine.

$8C      -- WPL CURRENT CHARACTER STASH

        Remembers the current WPL character being
        evaluated. Used to search for a "=" string
        assignment and to end on a carriage return.

$8E      -- STRING $A-$D POINTER

        Inits to $A =$00, $B=$40, $C=$80, $D=$C0 to point
        to correct string start in $1E00 buffer.
        Incremented after each string character access.

```



## Listing C.7—cont.

## \$90,91 -- PRINTER POINTER TO TEXT FILE

Initiates to \$0801, the start of the textfile. Used to get one character at a time from LOFILE for printing. On an aborting [esc], saves final position to \$98,99.

## \$92 -- WPL SUBROUTINE STACK POINTER

Points to address pairs in the WPL stack work file at \$1D00-1D40. Initiates to zero. A six bit pointer, limited to 64 values, and pointing to one of 32 possible pairs of WPL return addresses.

## \$94,95 -- WORD DELETION POINTER

Used by [W] and [X] to access deletion buffer at \$1800-1BFF. Goes round and round. Advances each time a character is added to buffer and retards each time a character is removed.

## \$96,97 -- TAB COUNTER

Holds the POSition from the last carriage return. Used by tab routines and the status line.

## \$98,99 -- STATIC CURSOR POINTER

Holds the static cursor for [Y] split screen. Holds the return display cursor for [S]. Holds initial search pointer for [F]. Remembers the start of file for [P]. Used to calculate the relative direction of character movement when doing a HICURS->LOCURS or LOCURS->HICURS move.

## \$9A-9B -- MEMORY LEFT STASH

Subtracts HICURS-LOCURS to find out how much memory is left. Used by status line.

## \$9E-9F -- PRINT DESTINATION POINTER

Holds the starting address of a printer routine. Normally an interface card "adjusted" \$C100 for slot one, \$C200 for slot two etc. Set to \$4397 for print to disk, or to \$4415 if print to screen.

## \$A0-A1 -- WPL PROGRAM COUNTER

Points to the next character in the WPL program starting at \$1000-17FF. Initiates to \$1000 and is saved to the WPL stack on a subroutine call. Restored from the stack on a subroutine return. Set to new value on [P]-GO command.

## \$A2 -- MULTI-USE LOCAL STASH

The search pointer for [F]ind. the first character in a PP command save. The searching delimiter in [L]oad.

## Listing C.7—cont.

```

$A3  -- MULTI-USE LOCAL STASH

      A replacement flag for [F]ind, with $00 being
      no replacement, and $FF allowing replacement.
      The second character save hold in a PP command
      save. The second delimiter in [F]ind and [L]oad.

$A4  -- END DELIMITER

      Pointer to the third delimiter during [L]oad.

$A6  -- VERTICAL SCREEN POSITION

      Holds the current vertical screen position.

$A7  -- HORIZONTAL SCREEN POSITION

      Holds the curent horizontal screen position.
      Note that location $24 is not used for this
      purpose.

$A8  -- ARITHMETIC MODE FLAG

      If $FF, absolute arithmetic during [P] values.
      If $2D, relative negative arithmetic (ASCII "-")
      If $2B, relative positive arithmetic (ASCII "+")
      Negative values are 2's complemented during entry
      so that a simple add-only routine can handle both
      "+" and "-" relative calculations.

$AA-AB -- PROMPT POINTER

      Points to the text prompts needed by the bottom
      screen window on certain [ ] commands.

$AC  -- SWALLOW BUFFER POINTER

      Points to the last uded location in the single
      character swallow buffer at $0300-037F. MSB
      is ignored, forcing pointer round and round on
      128 values. Pointer gets incremented on single
      character insertions and decremented on single
      character deletions.

$AD  -- STRING SOURCE FLAG

      A $00 here gets new strings from keyboard. An
      $FF uses the existing string source.

$AE-AF -- AUXILIARY UTILITY POINTER

      Used by copy from memory to scan text file.
      Finds start of present screen line in line start
      routine. Scanning pointer when using delimiters
      with [S]ave. A filling pointer for the sector
      buffer when using delimiters. An access pointer
      when clearing old high ASCII start-of-line marks.
      A pointer to set new start-of-line marks.

$B0  -- FILENAME SOURCE FLAG

      If $00, get a new filename. IF $FF, use the
      old "=" filename.

```

## Listing C.7—cont.

```

$B1  -- SCREEN RIGHT MARGIN

      Inits to 79, but is settable from 0-240.  [Z]
      mode with whole word breaks must be used if >79.

$B2  -- SCREEN LEFT MARGIN

      Inits to left margin print value on [A].
      Horizontal scrolling activated if RM-LM > 78.

$B4  -- STARTUP FLAG

      Sets on cold boot to $FF, allowing an attempt
      at running a STARTUP program with no error
      message if not found.  Resets to $40 after first
      startup attempt.

$B5  -- CURSOR SYMBOL STASH

      A $20 value here means to use a white box cursor
      that is software flashible.  A $00 value means
      an "off" cursor.

$B7  -- SPACE LEFT ON LINE STASH

      Inits to LM.  PM gets adjusted if the first
      line in paragraph.  The actual length of the
      printable line, always rounded off to whole
      words.

$B8  -- PRINTER ENABLE FLAG

      An $00 means that the printer is off or that the
      screen is to be the print destination.  A value
      of $FF means an active printer.

$B9  -- HIFILE/LOFILE FLAG

      Set by the screen pointer switch and used by
      vertical cursor routines.  $00 = LOFILE.
      $FF = HIFILE.

$BA,BB -- REFORMATTING POINTER

      Used when reformatting the textfile to mark start
      of screen lines.  The final character in any
      screen line is set to high ASCII, while all other
      characters are cleared to low ASCII.

$BC  -- TAB-OVER COUNTER

      Adds needed number of spaces to screen when using
      the tab-over feature.

$BD  -- MEMORY PAGE COUNTER

      Used when transferring textfile image to a 512
      byte ProDOS sector buffer so that delimiters can
      be searched.

$BE,BF -- RUNNING PAGE NUMBER COUNTER

      Inits to PN and is incremented on each new page
      as it is printed.

```

## Listing C.7—cont.

```

$C0  -- MULTI USE STASH

      By itself, a print enable stash used with EP0
      and EP1. Holds the conditional value for a
      conditional formfeed. Holds baud rate and modem
      values during modem setup. Hex stash when used
      with $C2 and $C3.

$C1  -- HEXADECIMAL INTERMEDIATE BYTE

      See below.

$C2  -- MULTI-USE LOCAL STASH

      Accumulator save for type-ahead buffer. Local
      character hold in several routines. Screen start
      pointer variable. Glossary match character hold.
      Error number stash for WPL error processor. Hex
      stash when used with $C0 and $C2.

$C0-C2 -- HEXADECIMAL STASH

      Holds the hex source for conversion from hex
      to decimal and the hex result for decimal to
      hex. LSB is in $C0, intermediate byte in $C1
      and high byte is in $C2. If less than a 24 bit
      conversion, then $C2 is zero. If less than a
      16 bit conversion, then $C1 is zero. 24 bit
      range is needed for ProDOS catalog and append
      calculations.

$C4  -- CASE FLAG

      If $00, normal mixed upper and lower case.
      If $80, all upper case.
      If $C0, all lower case

$C5  -- LOCAL Y-REGISTER SAVE

      Used by many routines for low level hold of the
      Y register. Also a local variable in the screen
      line reformatting.

$C6  -- LOCAL X-REGISTER SAVE

      Used by many routines for low level hold of the
      X register. Also a local variable in the screen
      line reformatting.

$C7  -- LOCAL ACCUMULATOR SAVE

      Used by many routines for low level hold of the
      accumulator. Cursor motion direction flag with
      $00 = to LOFILE and $FF = to HIFILE. Match
      character hold for [F]ind and [L]oad. Right
      margin hold for screen line formatter. Character
      hold for screen update. Padding counter for
      fill justify. One busy mother.

$C8  -- SCREEN CENTERING STASH

      Used to find the line with the cursor on the
      screen display. Inits to six for a split screen
      and twelve for a full screen. Adjusted -3
      if page/position display is active.

```

## Listing C.7—cont.

```

$C9  -- ANY LENGTH FLAG

      If $FF, then continue an any length search,
      ignoring intermediate characters.  If $00,
      do not use any length feature.

$CA  -- HORIZONTAL SCROLLING TRIGGER

      Inits to 68, the point on the 80 character screen
      where right horizontal scrolling is needed.  Used
      to calculate screen start pointer.

$CB  -- SCREEN RIGHT MARGIN

      Inits to screen left margin plus 79, the maximum
      width normally displayable on screen.  Horizontal
      scrolling will trip if you get near the left or
      right margin, adjusting this value in the process.

$CD  -- REFORMATTING FLAG

      If $00, the high ASCII start of line markers in
      the textfile are all correct.  If $FF, then a
      screen reformatting is needed.

$CE  -- SCREEN SOURCE FLAG

      An $00 value uses LOFILE to get characters for
      the screen up to and including the cursed
      character position.  a $FF value uses HIFILE as
      a source for all characters from just past the
      cursor to the end of the screen.

$CF  -- DATA DIRECTION FLAG

      If $00, the data direction is "<".
      If $FF, the data direction is ">".

$DE  -- SLOT NUMBER STASH

      Holds the print destination *16.  Thus slot 2
      becomes the "2" in $C200, etc.

$DF  -- WPL AND GLOSSARY ACTIVITY FLAG

      If $00, neither WPL nor the glossary are active.
      If the N bit (MSB) is set, then WPL is active.
      If the V bit (MSB-1) is set, then the glossary
      is active.  If both flags are set, then WPL is
      making use of the glossary.

$EO  -- UNDERLINE FLAG

      If $00, then no underlining.  If $FF then
      underline.  Underlining is done by individually
      backspacing and underlining each character.
      The UT character toggles this flag.  Status is
      held during TL, BL, or a footnote.  NOTE: It
      is far better to underline with imbedded print
      constants.

```



## Listing C.7—cont.

```
$E1  -- WRAPAROUND FLAG

      A $00 value gives you a normal display of whole
      words only, while $FF breaks the words as needed
      at the right margin. You must break whole words
      if the screen is wider than 79 characters.

$E2  -- THE "A" FLAG

      Two uses. Do nothing if $00. APPEND on [S]ave
      if $FF. Continue for ALL occurrence on [L]oad
      or [S]ave if $FF.

$E5  -- DATA LINE TOGGLE FLAG

      If $00, then no status display at all.
      If $80, then the usual Mem: Len:, etc. display.
      If $C0 then the tab display.

$E6  -- DELIMITER CHARACTER STASH

      Holds the delimited character in use for
      matching. Used by [S]ave, TL, BL, and special
      delimiter calculations.

$E7  -- WPL CONTINUE FLAG (?)

      Set to $00 by [P]-GO and [P]-DO. Read by main
      word processor loop and apparently used as a
      branch always. No obvious use; possibly a
      debugging hook or future WPL extension.

$E8  -- CASE CHANGE FLAG

      If $00, use mixed upper and lower case. If $FF,
      use the case held in the $C4 case flag. This
      seems to be a leftover from two versions back,
      as it is never referenced.

$E9  -- FILENAME LENGTH COUNT

      Holds the number of characters in the filename.
      Used by ProDOS access, and to find "\" for screen
      only load, or "+" to append during [S]ave.

$EA  -- WILDCARD SPECIAL DELIMITER

      Used to hold the match character for an "any
      character" search. Holds an $03 when using the
      default "/" delimiter which does not allow
      wildcards. The wildcard character is the
      ASCII value of the special delimiter plus
      three.

$EB  -- CARRIAGE RETURN SPECIAL DELIMITER

      Used to hold the match character sitting in for
      a carriage return during a search. Holds an $02
      when using the default "/" delimiter which does
      not allow carriage returns. The carriage return
      character is the ASCII value of the special
      delimiter plus two.
```



## Listing C.7—cont.

**\$EC -- ANY LENGTH SPECIAL DELIMITER**

Used to hold the match character for an "any length" string during a search. Holds an \$01 when using the default "/" delimiter which does not allow any length matches. The any length character is the ASCII value of the special delimiter plus one.

EXAMPLE: "<", "=", ">", and "?" are four ASCII characters in sequence. If "<" is the special delimiter, then "=" will be the any length symbol, ">" will be the carriage return symbol, and "?" will be the wildcard.

**\$ED -- MYSTERY FLAG**

Its still with us, kiddies. Zeroed at entrance to word processing code. Not otherwise referenced. Apparently another leftover.

**\$EE -- [W] AND [X] OVERFLOW FLAG**

If \$00, all is well. If \$FF, an attempt has been made to save more than 2048 characters to the deletion buffer.

**\$EF,F0 -- [W] AND [X] OVERFLOW COUNTER**

Counts the characters going into the deletion buffer and sets \$EE flag if an attempt is made to save more than 2048 characters at once.

**\$F2 -- TYPE-AHEAD BUFFER EMPTIER**

Points to the last used character in the type-ahead buffer at \$1D40-1D7F. Limited to six bits for 64 possible round-and-round locations. Gets "behind" \$F3 when busy. Increments on each character use.

**\$F3 -- TYPE-AHEAD BUFFER FILLER**

Points to the next available character location in the type-ahead buffer at \$1D40-\$1D7F. Limited to six bits for 64 possible round-and-round locations. Increments on each keystroke that cannot be immediately used.

**\$F4 -- BUSY PROMPT**

A \$20 or a white square on the status line when not busy. A \$2A or an inverse "\*" on the status line when busy.

**\$F5 -- REPLACE MODE FLAG**

If \$00, use normal entry mode. If \$FF, overwrite cursed character. Toggled by [R] and reset by just about all cursor commands.

## Listing C.7—cont.

**\$F6 -- STRING \$A-\$D ACTIVITY FLAG**

Inhibits WPL interpreter when \$A-\$D strings are being processed. A \$00 means normal WPL use; use; a \$FF means a string is being processed.

**\$F7 -- SCREEN DISPLAY FLAG**

A \$00 value means to display to the screen. An \$FF value means to not display to the screen. Controlled by [P]-ND and [P]-YD.

**\$F8 -- SPLIT SCREEN FLAG**

If the N bit (MSB) is clear, then the screen is unsplit. If the N flag is set, then the screen is split. If the V bit (MSB-1) is clear, then the upper half screen is active. If the V bit is set, then the lower half screen is active.

**\$F9,FA -- DELETION OR SPLIT SCREEN POINTER**

Used in several different and local ways. A pointer to the static half of a split screen display. A deletion pointer for word, character, or paragraph removal.

**\$FB -- APPLE KEY STASH**

Holds the open-apple and solid-apple keystrokes when the type ahead buffer is in use. A \$00 means neither key is down. A \$40 means the open-apple key is the only one down. A \$80 means the solid-apple key is down, while a \$C0 means both the open- and solid-apple keys are down.

**\$FC -- MACHINE ID**

Init to \$00 for "old" monitor IIe or IIc. Init to \$FF for "new" monitor IIe. Alters the stash addresses the modem parameters are passed to.

**\$FD -- SCREEN LOAD FLAG**

If \$00, load to file in the normal way.  
If \$FF, load to screen only.

**\$FE -- FOOTNOTE LINE COUNTER**

If \$00, no footnotes are in use. If \$FF, one footnote line remains. If \$FE, two footnote lines remain, \$FD three, etc.

**\$FF -- BOTTOM OF PAGE FLAG**

This flag init to \$00 and goes to \$FF whenever the body is completely printed, or on a formfeed taken. Footnotes and bottom line can then be entered.

## Listing C.8. AWD.SYS important entry points.

Important "F" version entry points include system level entry that accesses the entire program; command level entry that does control commands; WPL module entry that handles individual WPL commands; the auxiliary function access; the ProDOS command routines, the ProDOS machine language MLI links, and finally the often-used service subroutines.

Here they are:

```

-- system level entry --

$2000 - Cold start entry
$2003 - Print to screen (used by formatter)
$2006 - Get user response (used by formatter)
$2009 - ProDOS error processor
$20B4 - Warm restart entry

-- command level entry --

$2A26 - [Q] Unconditional delete
$4F70 - [A] Adjust margins
$27DF - [B] Cursor to start
$3D0D - [C] Case changer
$3D1E - [D] Data direction changer

$2808 - [E] Cursor to end
$3ACB - [F] Find, search and replace
$2A36 - [G] Glossary
$26EB - [H] Backspace left arrow
$32E8 - [I] Do actual tab

$270D - [J] Down arrow
$2715 - [K] Up arrow
$394A - [L] Load
$3CCD - [N] New
$4D6E - [O] DOS access

$417A - [P] Print/Program main entry
$2B7F - [Q] Auxiliary functions entry
$3D25 - [R] Replace mode toggle
$3648 - [S] Save
$3265 - [T] Tab set, clear, or purge

$26FC - [U] Frontspace, right arrow
$35B6 - [V] Verbatim mode toggle
$2961 - [W] Delete Word
$2961 - [X] Delete Paragraph
$34F9 - [Y] Split screen

$359E - [Z] Wraparound toggle
$2BC9 - [_] Position toggle

-- WPL modules --

$4714 - AS Assign String
$4603 - BL Bottom line
$4861 - CP Continue printing
$44DC - DO Run WPL program
$480E - EP Enable printer

```

## Listing C.8—cont.

```

$4636 - FF Formfeed
$465D - GO WPL unconditional jump
$46D6 - IN User keyboard response
$42BA - LS String Load
$431D - ND Screen display off

$4844 - NP Begin Printing
$46AD - PR Prompt to screen
$45F0 - QT End WPL program
$4305 - RT WPL subroutine return
$4321 - SC String Compare

$42E7 - SR WPL subroutine jump
$4608 - TL Top display line
$431E - YD Screen display on

-- auxiliary functions --

$2F84 - "A" Load Tab File
$2F5A - "B" Save Tab File
$2FDB - "C" Load Print File
$2FB1 - "D" Save Print File
$2EC6 - "E" Load Glossary

$2E74 - "F" Save Glossary
$35AF - "G" Toggle CR display
$3594 - "H" Toggle Data display
$2D16 - "I" Keyboard direct to printer
$2C97 - "J" Quit everything

-- ProDOS menu options --

$5124 - "A" Catalog
$5066 - "B" Rename
$50AE - "C" Lock
$50AA - "D" Unlock
$50FD - "E" Delete

$4FD7 - "F" List volumes On-line
$2C8F - "G" Create subdirectory
$510F - "H" Set prefix
$301D - "I" Initialize diskette
$4DAF - "J" Set modem interface

-- ProDOS interface links --

$2C83 - Get Prefix (used by catalog)
$2D05 - Quit (used by quit)
$2EAF - Write (used to save glossary)
$2F70 - Write (used to save tab file)
$2F9A - Read (used to load tab file)

$2FC7 - Write (used to save print file)
$2FF1 - Read (used to load print file)
$3045 - Read (used to init a volume)
$30E8 - Open (used by file creator)
$30F9 - Create (used by directory creator)

$3111 - Create (used by text file creator)
$324F - Read (used by text file loader)
$37CA - Write (used by text file saver)
$37EB - Close (used by text file loader)
$37F9 - Get EOF (used by append)

```

## Listing C.8--cont.

```

$3823 - Set EOF (used by append)
$38D9 - On Line (used by on-line)
$38F6 - Open (used by text file loader)
$391B - Set EOF (used by text file loader)
$439A - Write (single byte for PD8)

$509C - Rename (used by rename)
$50C8 - Get Attributes (used by lock/unlock)
$50D9 - Set Attributes (used by lock/unlock)
$5103 - Destroy (used by delete)
$5117 - Set Prefix (used by set prefix)

-- often used service subroutines --

$219D - Print character to screen link
$225F - Ring the ding dong
$2273 - Message link
$227E - Pick string source

$22B7 - Get key from type-ahead buffer
$2302 - KSWL entry (Get key immediate)
$2338 - Unflash screen character
$23AA - Flash screen character
$23FF - Print hex pair as ASCII

$24B7 - Scroll screen
$22F8 - BASH from HPOS
$24FA - BASH immediate
$2507 - Read modem
$2571 - Home cursor

$257C - Clear EOL from HPOS
$257E - Clear EOL immediate
$25C0 - Init LOFILE
$25CF - Mark LOFILE end
$25E8 - Init HIFILE

$25F7 - Mark HIFILE end
$2647 - Enter character to LOFILE
$269C - Move character LOFILE --> HIFILE
$26BB - Mark open end of LOFILE
$26C9 - Move character LOFILE <-- HIFILE

$27A5 - Update LOCURS pointer
$27C1 - Update HICURS pointer
$283F - Increment [W][X] pointer
$2848 - Increment WPL program counter
$284F - Test direction

$285B - Increment auxiliary utility pointer
$2862 - Increment LOCURS pointer
$2869 - Increment split pointer if < LOCURS
$2875 - Increment HICURS pointer
$287C - Increment screen pointer

$2883 - Increment main utility pointer
$288A - Increment printer pointer
$2891 - Decrement [W][X] pointer
$289B - Decrement [W][X] overflow counter
$28AC - Decrement LOCURS pointer

```



## Listing C.8—cont.

```
$28B7 - Decrement split pointer if > HICURS
$28C7 - Decrement HICURS
$28DD - Decrement main utility pointer
$28E8 - Set screen start pointer
$29F5 - Backspace routine

$2A26 - Unconditional delete character
$2AE0 - Reset glossary
$2ED7 - Get and hold filename (ref #0)
$2ED9 - Get and hold filename (ref #A)
$2EDF - Get filename or volume name

$2F0B - Set text file reference number
$2F13 - Set binary file reference number
$2F1B - Create and set binary file ref number
$2F36 - Hold filename
$3068 - Create and open text file

$30E0 - Open second file
$3126 - ProDOS error processor
$31EA - Set up disk or memory read
$3201 - Set up disk read
$324F - Read one disk sector to buffer

$33C6 - Main WP entry
$33F1 - Main WP service loop
$3491 - Process as command
$3543 - Unsplit screen
$35BD - Clear screen bottom and get string

$35C3 - Print filename
$360B - Clear and prompt screen bottom
$3638 - Clear screen bottom
$3727 - Add prefix to pathname
$3748 - Disk write setup

$377D - Save one sector to disk
$37DD - Close all files
$37E5 - Close one file
$37F9 - Append setup
$3811 - Append cleanup

$3831 - Process slot and drive
$38CD - Read slot, forcing 0-7
$38E6 - Read one sector from disk
$3929 - Process special delimiters
$3CAC - Move LOFILE <---> HIFILE

$3CF7 - Force upper case
$3D2C - Calculate POS
$3E58 - Print as inverse decimal
$3E69 - Fix mouse nest
$3E7D - Read character from LOFILE

$3EB2 - Reformat screen markers
$3EDB - Reformat screen line
$3F7A - Set auxiliary pointer to HICURS
$3F83 - Case changer routine
$3FA5 - Update screen

$40F6 - WPL error processor
$4144 - Return prompt routine
$43AC - Grab printer hooks
$44A1 - PRT values to screen
$44BB - Save old filename as =
```



## Listing C.8—cont.

```
$44C8 - Restore old filename from =
$453C - Keybuffer to pathname buffer move
$45B0 - WPL $A-$D string interpreter
$45F0 - Quit WPL cleanup
$4628 - Pad bottom lines

$4653 - Print space and carriage return
$49C7 - Save characters to line buffer
$4B1C - Pad left margin
$4BE2 - Print filter
$4C36 - Substitute NULL character

$4C41 - Print and delay
$4C76 - Pop subroutine
$4C78 - Stall for modem serial time
$4CDE - Character to printer link
$4CF6 - Clear screen via formfeed print

$4CF8 - Character to screen
$4EE5 - Print X and A as decimal ASCII
$4FA0 - Adjust margins setup
$4FBA - Adjust margins
$504D - Message to screen

$5124 - Catalog disk
$52A8 - Print hex as decimal
$52E0 - Print blanks to tab screen
$534B - Date and time to screen
$538D - Slash to screen

$5399 - Pathname to screen
$53B9 - Hex to decimal conversion
$53FD - Hex division, page/position
$540C - Decimal to hex conversion
$5470 - Hex X10 multiplier
```

## Listing C.9. Detailed script of AWD.SYS main program.

The AWD.SYS version of ProDOS Applewriter sits between \$2000 and \$5472 in main memory. The script that follows assumes an Apple IIc in 80 column mode, or else a 128K IIe having an extended 80 column card installed.

Note that a script of the internal file portions of this program appeared in Listing 7.3. Here is a module-by-module breakdown of the rest of the code:

\$2000-2002 -- COLD START ENTRY POINT

Jumps to actual cold start code at \$200C.

\$2003-2005 -- PRINT TO SCREEN ACCESS

Used by the formatter to print to screen. Vectors to \$2413.

\$2006-2008 -- GET USER LINE ACCESS

Used by the formatter to get user response. Vectors to \$227E.

\$2009-200B -- ProDOS ERROR PROCESSOR LINK

Vectors to error processing code at \$3126.

\$200C-20B1 -- COLD START MODULE

Init stack pointer to \$01FF. Set bottom of LOFILE to auxiliary memory \$0800. Clear all page zero locations above \$60. Save last slot and drive used. Zero the TL and BL buffers, the filename hold, the glossary, and WPL strings \$A-\$D. Fill the glossary with carriage returns. Zero the entire character swallow buffer. Route BRK, IRQ, and warm RST to \$20B4, the warm restart module. Set the power up byte. Read the machine id byte and test for an "old" or a "new" machine, saving the result in version flag \$FC. Disconnect 80 column hooks. NOTE: removes any third party card from slot three.

Install memory management code to page zero of main memory. Open to full screen window with normal text. Init LOFILE and RIFILE unless "zN" marker is set. If set, update LOCURS and HICURS pointers only, and mark with "zN" marker. Apparently lets another SYS program transfer control to Applewriter for processing. Set split screen pointer to LOCURS. Set cursor symbol to white box. Set right screen margin to 79 and right scroll trigger to 68. Put down main boot screen. Get PRT.SYS and TAB.SYS. Set screen margins. Fall through to warm restart module.

## Listing C.9—cont.

## \$20B4-20DB -- WARM RESTART MODULE

Zero modem activity flag. Set CSWL and KSWL to a "brick wall" RTS. KSWL stays there during entire program; CSWL gets diverted as needed to screen, printer, modem, or disk. Reset the stack pointer. Close all files. Re-install memory management code. Open to full screen and normal text. Use white block as cursor. Reset WPL via PQT. Reset the glossary stack. Jump to the main WPL service routine.

## \$20E0-2103 -- PICK INIT OR UPDATE

Used only by the cold start routine. If you enter this module without a high ASCII "zN" at \$B77E and \$B77F, then LOFILE and HIFILE get initied. If there is a "zN", then only the LOFILE and HIFILE pointers get updated. Apparently used for integrated program access or error recovery; normally inits HIFILE and LOFILE.

## \$2104-214E -- DISCONNECT SLOT 3 HOOKS

Used to save old 80-column connections used by a previously-run .SYS program. Old 80 column hooks are saved to \$214F-2151, and the hooks are removed from the ProDOS driver list. Later restored during quit. Not internally used by the word processor. Required of any ProDOS ".SYS" program. NOTE: makes any use of any third-party video or screen card very difficult.

## \$2152-215F -- INSTALL MEMORY MANAGEMENT CODE

Move the image of the memory management code to \$0100-013D. Memory management code is needed to access LOFILE and HIFILE in aux memory. All else is in main memory.

## \$215F-219C -- MEMORY MANAGEMENT CODE IMAGE

Gets moved and accessed on main memory page one. Here are the relocated addresses:

\$0100 - Read LOFILE by screen pointer.  
 \$0109 - Read LOFILE by LOCURS pointer.  
 \$0112 - Read HIFILE by HICURS pointer.  
 \$011B - Read LOFILE by printer pointer.  
 \$0124 - Read LOFILE by format pointer.  
 \$012D - Back format pointer up to start of screen line.

## \$219D-21AD -- GET AND PRINT CHARACTER TO SCREEN

Get the next character from the type-ahead buffer if in use, or directly if not. Reject if a \$00 NULL and get the next character. If WPL is off, save the character and print it to the screen.

## Listing C.9--cont.

## \$21AE-21BF -- SET STARTUP PATHNAME

Put the command DO STARTUP into the key buffer. Used to run a startup program on cold start or warm restart.

## \$21C0-21E4 -- TUTORIAL HELP TESTER

If open apple and "?" but no glossary activity, then load HELP 80 into key buffer, then DO HELP 80 via WPL.

## \$21E5-223B -- PUT DOWN FIRST BOOT SCREEN

Draw a hollow box on the screen. Home the cursor. Put down the boot screen image, using high ASCII values in the file as horizontal tab values to center the text in the box. Hollow box is printed by printing a line of inverse spaces, columns of inverse spaces at screen left and right, and then a final line of inverse spaces.

## \$2241-225E -- OPEN SCREEN WINDOW

Close all files. Set top of window to \$00. Set left of window to \$00. Set window bottom to \$18 = 24 lines. Access display page. Pick alternate character set. Use normal text. Clear screen if not WPL. Turn 80 column display on.

## \$225F-2272 -- DING DONG

Sound the two-tone alarm. Plays two plain old squarewave bursts back to back, the second lower in pitch. Calls itself for the first note.

## \$2273-227C -- PROMPT AND GET USER RESPONSE

Save the present HPOS, print the message pointed to by the X register high and the Accumulator low, then fall through to next module.

## \$227E-228C -- GET USER RESPONSE

If the string source flag is set, get the response from auxiliary memory where LS previously put it. If clear, get the response from the user. If WPL is active, substitute variables (x) through (z).

## \$228D-22B1 -- FILL TYPE-AHEAD BUFFER

Save the old cursor symbol. Use no current cursor. Put a busy signal "\*" into busy flag. Get the key from user. Advance the filler pointer \$F3. Save key to type-ahead buffer at \$1D40. If an Apple key was pressed, save it to the Apple buffer starting at \$1FC0. Restore the old cursor symbol.

## Listing C.9—cont.

## \$22B2-22B5 -- TYPE-AHEAD CHECK

Compare the type-ahead filler \$F3 against the emptier \$F2. If equal, set the carry flag. Equal means that the type-ahead buffer is not currently needed.

## \$22B7-22D2 -- EMPTY TYPE-AHEAD BUFFER

Save registers. Do type-ahead check. If not busy, fall through to next module. If in use, Increment the F2 emptier, forcing it to one of 64 values. Get stored Apple key and save to \$FB. Get the keystroke and hold in the accumulator. Restore registers.

## \$22D3-2301 -- NO LONGER BUSY

Save registers. If no status lines, fall through to next module. If a status line is needed, turn busy flag \$F4 off. Find active status line. Bash address. Poke unbusy signal to status line. Rebash current HPOS. Then fall through.

## \$2302-2323 -- GETKEY IMMEDIATE LINK

Save registers if not already saved. Get key. If glossary is not yet active and open apple is pressed, activate the glossary and read first glossary character. Restore registers. Access page one of main memory.

## \$2324-2334 -- PICK KEYSTROKE SOURCE

Clear modem activity flag \$23A9. If the glossary is active, get key from glossary. If WPL is active, get key from the WPL interpreter. If the modem is active, get key from modem buffer. If all else fails, fall through to keyscanner and get the key from the user.

## \$2348-237B -- KEY SCANNER

If modem is not active, set up medium time delay keyboard interrogation loop. Read the keyboard, falling through on a pressed key. If modem activity, abort immediately. Time out for half cursor flash period, then curse screen. Repeat until key is pressed or until modem is active. Note that the cursed character is flashed by delaying, switching to inverse, delaying, switching to normal, and so on.

## \$237B-2395 -- GOT KEY FROM USER CLEANUP

Clear modem activity flag. Clear cursed character back to normal. Save apple keys to \$FB. Read keyboard to accumulator. reset keyboard strobe.



## Listing C.9—cont.

## \$2396-23A8 -- GOT KEY FROM MODEM CLEANUP

Set modem activity flag. Clear cursed character back to normal. Clear apple key hold \$FB. Read modem to accumulator.

## \$23AA-23D7 -- FLASH SCREEN CHARACTER

Abort if invisible cursor or WPL is active. Get HPOS and divide by two to pick even or odd screen. Turn on even or odd screen. Get character. Change from low to high ASCII or vice versa. If a hit on the mouse nest, pick true inverse character. Save cursed character. Reset to main memory.

## \$23D7-23EC -- WRITE CHARACTER TO SCREEN IMMEDIATE

Use HPOS or Y register. Divide position by two and pick even or odd screen. Turn on even or odd screen. Store character to screen.

## \$23ED-23FE -- READ CHARACTER FROM SCREEN

Use HPOS. Divide by two and pick even or odd screen. Turn on even or odd screen. Load accumulator from screen.

## \$23FF-2412 -- PRINT HEX PAIR AS ASCII

Save the character. Shift high byte into low byte. Force numeric. If 10-16, then change to letter by adding \$06. Change to ASCII by adding \$30. Fall through to print as subroutine. Get character back. Mask the low byte, change to ASCII and fall through again.

## \$2413-243A -- PRINT TO SCREEN SETUP

Increment tab-over counter. If a carriage return, reset tab over counter. Test catalog-to-file flag \$BC. If set, jump to print to file routine. If clear, save registers, put character on screen, check modem, affirm main memory, and restore registers.

## \$243B-249F -- FILTER SCREEN CHARACTERS

Force high ASCII (normal text). Test modem flag. If slave, force low ASCII and print to modem instead of screen. If a form feed \$8C or a form separator \$9C, then clear the screen. If a bell \$87, ring the ding dong. If a carriage return, check the keyboard for a [S] scrolling stop. If stopped, read the keyboard until any new key is pressed, then reset the key strobe. Continuing the filter action, if a delete or a backspace, decrement HPOS. If the horizontal position underflows, decrement the vertical position, BASH the new base address, and set HPOS to decimal 79. This backs you up to the rightmost slot on the previous line. If a \$1E range separator,



## Listing C.9—cont.

substitute an inverse "A" to simulate the open apple key.

\$24D3-24F7 -- PUT CHARACTER ON SCREEN

Enter character to screen immediate. Add one to HPOS, checking for line overflow. If an overflow, reset HPOS to zero and add one to VPOS. If no overflow, BASH new VPOS and exit. If an overflow, fall through to screen scrolling routine.

\$24B7-24F7 -- VERTICAL SCROLL

Calculate screen position and save to screen scroll pointer. Bash screen line address. Move present screen line to screen scroll pointer one line higher, repeating for the needed number of lines to scroll the window. Move 40 even characters first in main memory, followed by 40 odd characters in aux memory. Check the modem after each half-line move. When finished, clear the bottom screen line and fall through to BASH its address.

\$24F8-2506 -- BASH BASE ADDRESS

Double VPOS pointer and use it to get the screen base address low for \$28 and the screen base address high for \$29 from the BASH Table. Note that the table lookup method is faster than direct calculation.

\$2507-254C -- READ MODEM TO MODEM BUFFER

Abort if slot zero, modem set in process, not slave mode, or if slot does not contain modem firmware. Get character from modem, and save to modem buffer \$1F40. Advance filler pointer \$254D.

\$2531-254C -- READ MODEM BUFFER

Abort if slot zero, modem set in process, not slave mode, if slot does not contain modem firmware, or if buffer is empty. Set the modem flag, increment the emptier pointer, and fill accumulator with character from modem buffer.

\$254F-2552 -- CSWL LINK

Jump to the character hooks, usually a "brick wall" RTS, the screen, a printer, a modem, or a disk driver under pd8.

\$2552-2559 -- CLEAR SCREEN

Set HPOS to zero and VPOS to top of window being cleared. Fall through to clear window routine.

## Listing C.9—cont.

\$255A-256E -- CLEAR WINDOW

Load Y with HPOS and A with VPOS. Bash the screen base address. Clear to end of line from Y. Repeat until window bottom.

\$2571-257B -- HOME CURSOR

Set HPOS to zero and VPOS to top of active screen window. Bash VPOS.

\$257C-258E -- CLEAR TO END OF SCREEN LINE

Entry point picks HPOS or Y as starting character. Hold Y register. Print space to screen. Restore Y register. Continue until 80th column.

\$25C0-25CE -- INIT LOFILE

Mark start of LOFILE with \$FF. Mark next location with \$00.

\$25CF-25E7 -- MARK LOFILE START

Set LOFILE pointer to \$0800. Write \$FF to auxiliary memory. Increment LOCURS.

\$25E8-25F6 -- INIT HIFILE

Mark end of HIFILE with \$FF. Mark previous location with \$00.

\$25F7-260A -- MARK HIFILE END

Set BIFILE pointer to \$BEFF. Write \$FF to auxiliary memory. Decrement HICURS.

\$260E-2619 -- CHARACTER TO FILE LINK

Save registers. Do character to file via subroutine. Restore registers.

\$261A-2622 -- CHARACTER TO SCREEN LINK

Save registers. Do character to screen via subroutine. Restore registers.

\$2627-2646 -- CHARACTER TO SCREEN AND FILE

Bypass screen display if WPL or slave modem. Ignore if character is beyond right screen margin. Set to normal text, high ASCII. Put character on screen if not a carriage return. Increment HPOS. Reset case change flag \$C4. Fall through to next module.

\$2647-2663 -- CHARACTER TO FILE SETUP

Save character in character hold \$C2. Test screen only flag \$FD. If screen only, print to screen and exit. If to file, calculate amount of remaining memory by subtracting LOCURS from HICURS. If no memory left, exit via out of memory error and cleanup. If enough memory, fall through to next module.

## Listing C.9—cont.

## \$2663-269B -- CHARACTER TO LOFILE IMMEDIATE

Force low ASCII. Bypass if \$00 or \$7F, disallowing NULL or DELETE. Set write to auxiliary RAM flag. If in the replace mode under [R], get the cursed character to be overwritten. If the character is not a carriage return or a file limit, then increment HICURS, and zero the old HICURS character, thus setting up an overwrite. For either normal or replace, next save the new character to the top of LOFILE, increment the [Y] pointer and the LOCURS pointer, mark the open end of LOCURS with an \$00, and restore the character to the accumulator.

## \$269C-26BA -- MOVE CHARACTER LOFILE ---&gt; HIFILE

Used to backspace. Decrement LOCURS and get character from LOFILE, testing for start of file. If no characters left, init LOFILE and exit. Read LOFILE character. Test case change flag \$C4 and change case if needed. Store character to bottom end of HIFILE in auxiliary memory. Decrement the HICURS pointer. Write \$00 to the open ends of both LOFILE and HIFILE. Exit with moved character in the accumulator.

## \$26BB-26C8 -- ZERO OPEN LOFILE AND HIFILE ENDS

Mark the high "open" end of LOFILE and the Low "open" end of HIFILE with \$00 markers. Preserve the accumulator while doing this.

## \$26C9-26E8 -- MOVE CHARACTER LOFILE &lt;--- HIFILE

Increment HICURS and get the character there. If at high end of HIFILE, then init HICURS and exit. Otherwise, check the case flag at \$C4 and change the case if needed. Store the character to the top end of LOFILE. Increment LOCURS. Mark the open ends of LOFILE and HIFILE with \$00 markers. Used to frontspace.

## \$26EB-26FB -- BACKSPACE SETUP

Check for an open apple key down. If down exit via save to swallow buffer routine. Fill Y with a space to mark stop of express cursor motions. If the open apple key is up, set up common cursor motion routine to move LOFILE <--- HIFILE, then do it.

## \$26FC-270C -- FRONTSPACE SETUP

Check for an open apple key down. If down exit via restore from swallow buffer routine. Fill Y with a space to mark stop of express cursor motions. If the open apple key is up, set up common cursor motion routine to move LOFILE ---> HIFILE, then do it.

## Listing C.9—cont.

## \$270D-2714 -- CURSOR DOWN SETUP

Set up common cursor motion routine to do cursor down routine, then do it.

## \$2715-271A -- CURSOR UP SETUP

Set up common cursor motion routine to do cursor up routine, then do it.

## \$271B-2743 -- COMMON CURSOR MOTION ROUTINE

Force feed the selected routine into the jump starting at \$2743:

```
Backspace - $269C (LOFILE <--- HIFILE)
Frontspace - $26C9 (LOFILE ---> HIFILE)
Cursor Down - $2769 (Cursor Down code)
Cursor Up - $2746 (Cursor Up code)
```

Save the express motion match character to \$5E30, using space for by-words horizontally and \$00 for end-of-file vertical motions. Then check the solid apple key. If up, then do the motion once via the forced jump. If down, set vertical motions to 22 lines if full screen and 10 lines if split. Do one needed motion. Repeat until a space or end of file horizontally, or until the number of lines or end of file vertically. Note this is self-modifying code.

## \$2746-2767 -- CURSOR UP IMMEDIATE

Reset the case change flag \$C4 to \$00. Test the wraparound flag and bypass on wraparound. Move line of characters LOFILE to HIFILE, stopping on the high ASCII start-of-line marker, and quitting if beginning of file. Move first character in line from LOFILE to HIFILE.

## \$2769-2788 -- CURSOR DOWN IMMEDIATE

Reset the case change flag \$C4 to \$00. Test the wraparound flag and bypass on wraparound. Move line of characters HIFILE to LOFILE, stopping on the high ASCII start-of-line marker, and quitting if end of file. Move first character in line from HIFILE to LOFILE.

## \$278A-27A4 -- WRAPAROUND CURSOR UP/DOWN

If in wraparound mode, whole screen lines are moved without regard to start-of-line high ASCII markers. Reset the case flag \$C4. Pick direction, LOFILE ---> HIFILE if cursor up and LOFILE <--- HIFILE if cursor down. Move up to 80 characters, aborting on a carriage return. Note that wraparound is only allowed when the right screen margin is less than 79.



## Listing C.9—cont.

## \$271B-2743 -- COMMON CURSOR MOTION ROUTINE

Force feed the selected routine into the jump starting at \$2743:

```
Backspace   - $269C  (LOFILE <--- HIFILE)
Frontspace  - $26C9  (LOFILE ---> HIFILE)
Cursor Down - $2769  (Cursor Down code)
Cursor Up   - $2746  (Cursor Up code)
```

Save the express motion match character to \$5E30, using space for by-words horizontally and \$00 for end-of-file vertical motions. Then check the solid apple key. If up, then do the motion once via the forced jump. If down, set vertical motions to 22 lines if full screen and 10 lines if split. Do one needed motion. Repeat until a space or end of file horizontally, or until the number of lines or end of file vertically. Note this is self-modifying code.

## \$2746-2767 -- CURSOR UP IMMEDIATE

Reset the case change flag \$C4 to \$00. Test the wraparound flag and bypass on wraparound. Move line of characters LOFILE to HIFILE, stopping on the high ASCII start-of-line marker, and quitting if beginning of file. Move first character in line from LOFILE to HIFILE.

## \$2769-2788 -- CURSOR DOWN IMMEDIATE

Reset the case change flag \$C4 to \$00. Test the wraparound flag and bypass on wraparound. Move line of characters HIFILE to LOFILE, stopping on the high ASCII start-of-line marker, and quitting if end of file. Move first character in line from HIFILE to LOFILE.

## \$278A-27A4 -- WRAPAROUND CURSOR UP/DOWN

If in wraparound mode, whole screen lines are moved without regard to start-of-line high ASCII markers. Reset the case flag \$C4. Pick direction, LOFILE ---> HIFILE if cursor up and LOFILE <--- HIFILE if cursor down. Move up to 80 characters, aborting on a carriage return. Note that wraparound is only allowed when the right screen margin is less than 79.

## \$27A5-27C0 -- UPDATE LOCURS POINTER

Affirm a \$FF at start of LOFILE. Set LOCURS pointer to \$0800 start of LOFILE and scan upwards until a top-of-LOFILE \$00 marker is found. Hold address in LOCURS \$84,85.

## \$27C1-27DE -- UPDATE HICURS POINTER

Affirm a \$FF and end of HIFILE. Set HICURS pointer to \$BEFF end of HIFILE and scan downwards until a bottom-of-HIFILE \$00 marker is found. Hold address in HICURS \$86,87.

## Listing C.9—cont.

\$27DF-2807 -- CURSOR TO BEGINNING

Set data direction flag \$CF to ">". Begin moving characters from LOFILE to HIFILE, until the \$FF start-of-LOFILE \$FF marker is found. Update both LOCURS and HICURS.

\$2808-283C -- CURSOR TO END

Set data direction flag \$CF to "<". Begin moving characters from HIFILE to LOFILE, until the \$FF end-of-HIFILE \$FF marker is found. Update both LOCURS and HICURS.

\$283F-2847 -- INCREMENT [W] [X] POINTER

Increment the word and paragraph deletion pointer \$94, 95. Then decrement the word and paragraph overflow counter \$EE, EF. Set overflow flag \$EE if > 2048 characters. Then range check the [W] [X] pointer to keep it on memory pages \$1800-\$1BFF.

\$284F-285A -- TEST [Y] POINTER

Compare the split screen [Y] pointer against LOCURS, clearing the carry flag if [Y] is less than LOCURS.

\$285B-2861 -- INCREMENT AUXILIARY UTILITY POINTER

Add one to auxiliary pointer pair \$AE, AF.

\$2862-2868 -- INCREMENT LOCURS

Increment LOFILE cursor pointer pair \$84,85.

\$2869-2874 -- INCREMENT [Y] POINTER IF < LOCURS

Test the split screen [Y] pointer. Then increment it only if it is less than LOCURS.

\$2875-287B -- INCREMENT HICURS

Increment HIFILE cursor pointer pair \$86,87.

\$287C-2882 -- INCREMENT SCREEN POINTER

Increment screen pointer pair \$88,89.

\$2883-2889 -- INCREMENT MAIN UTILITY POINTER

Increment general use pointer pair \$80,81.

\$288A-2890 -- INCREMENT PRINTER POINTER

Increment printer pointer pair \$90,91.

\$2891-28AB -- DECREMENT [W] [X] POINTER

Decrement word and paragraph deletion pointer pair \$94,95. Then decrement the deletion overflow counter \$EF,F0. Set overflow flag \$EE if > decimal 2048 characters. Then range check the [W] [X] pointer to keep it on memory pages \$1800-1BFF.



## Listing C.9—cont.

\$28AC-28C6 -- DECREMENT LOCURS

Decrement LOFILE cursor pointer pair \$84,85.

\$28B7-28C6 -- DECREMENT [Y] POINTER IF < LOCURS

Test the split screen [Y] pointer. Then increment it only if it is less than LOCURS.

\$28C7-28D1 -- DECREMENT HICURS

Increment HIFILE cursor pointer pair \$86,87.

\$28D2-28DC -- DECREMENT SCREEN POINTER

Decrement screen pointer pair \$88,89.  
Apparently never used in this version.

\$28DD-28E7 -- DECREMENT MAIN UTILITY POINTER

Decrement general use pointer pair \$80,81.

\$28E8-2953 -- SET SCREEN START POINTER

Calculates the start of the screen line twelve lines above screen center for full screen and six lines above center for split screen. Aborts if it gets to the beginning of the file. Works by starting with the auxiliary utility pointer equal to LOCURS, and backing up to the start of each screen line, as set by a high ASCII marker. On the first trip through, the horizontal scrolling module below is used to find a suitable on-screen left margin. The keyboard is sampled once each on-screen line, and any pressed keys are held in the type ahead buffer. When the calculation is complete, the screen start pointer at \$88,89 is updated and then incremented by one.

\$2903-2935 -- HORIZONTAL SCROLL SET

Calculates a suitable left screen display margin on the first trip through the screen start set module. If the right margin is 78 or less, the left screen start margin \$B2 is set to zero. On a right margin above 79 characters, if the cursor is left of 11 characters to the right of the old left margin, then the left screen margin is set to the present cursor position minus eleven, thus doing a horizontal scroll left. If the cursor is between 11 and 68 characters to the right of the old screen left margin, then no scrolling is needed, and the old left margin is reused. If the cursor is more than 68 characters to the right of the old left margin, then a horizontal scroll right is done, setting the left margin to 68 characters left of the cursor position. The final calculated left margin is then saved to \$B2.

## Listing C.9—cont.

\$2954-2960 -- RANGE CHECK [W] [X] POINTER

The high byte of the word and paragraph is logically screwed around with to keep it pointing to pages \$1800-\$1BFF, inside the deletion buffer.

\$2961-2987 -- WORD AND PARAGRAPH DELETION SETUP

Reset the [W] [X] overflow flag \$EE. Init the overflow counter \$EE,EF for a 2048 character count. Store a \$20 space if [W] or an \$0D charage return if [X] to the stop-on-match stash at \$02. Test the data direction register, and pick insertion or deletion. Begin loop inserting or removing characters from file to or from the deletion buffer, and aborting on a match character, the start of LOFILE, the end of HIFILE, a carriage return, or overload. Test overflow flag, sounding alarm on overflow. Exit by reformatting the screen.

\$29B1-29B8 -- [W] [X] SINGLE INSERTION

Read a character from the word and paragraph deletion counter. Increment the [W] [X] counter and decrement the overload counter. Output the character to the textfile.

\$29B9-29E4 -- [W] [X] SINGLE DELETION

Decrement the split screen pointer. Test the solid apple key for copy rather than grab. If copy, get character, then move character from LOFILE to HIFILE. If at start of LOFILE, then re-init LOFILE. Decrement [W] [X] counter, and decrement the overflow counter, setting \$EE on an overflow. Save the character to the deletion buffer. If grab, decrement LOCURS, then read LOFILE. If at start of LOFILE, then re-init LOFILE. Decrement [W] [X] counter, and then decrement the overflow counter, setting \$EE on an overflow. Save the character to the deletion buffer. Mark the open end of LOFILE with an \$00 marker.

\$29E5-29F4 -- BARF SINGLE CHARACTER

Decrement the swallow buffer pointer, while restricting it to 128 characters maximum, round and round. Read the character from the swallow buffer \$0300. Save the character to the textfile at LOCURS. Reformat screen markers.

\$29F5-2A19 -- SWALLOW SINGLE CHARACTER

Decrement the split [Y] pointer. Decrement LOCURS. Force swallow buffer pointer to 128 possible values. Get character from LOFILE at LOCURS. If at bottom of LOFILE, restore LOFILE start. Save the character to the swallow buffer. Increment the swallow pointer \$AC. Mark over the character

## Listing C.9—cont.

```

                                at LOCURS with an $00 open-end-of-file
                                mark. Reformat the screen markers.

$2A1A-2A25 -- RESTORE LOFILE START

                                Increment split screen pointer. Init LOFILE.
                                Save accumulator. Reformat screen markers.
                                Restore accumulator from stack.

$2A26-2A35 -- DELETE CHARACTER UNCONDITIONALLY

                                Decrement LOCURS. Read character at LOCURS.
                                if at LOFILE start, init LOFILE and reformat
                                screen margins. If not, mark over the
                                character at LOCURS with an $00 open-end-of-
                                file mark. Reformat the screen markers.

$2A36-2A53 -- GLOSSARY SETUP

                                Clear and prompt screen bottom. Get glossary
                                command. Ignore if NULL. If a carriage
                                return, then exit. If define "?", then
                                do define module. If purge, then mark start
                                of glossary at $0800 in main memory with $00
                                and exit. If any other character, then do
                                glossary entry module.

$2A54-2AA9 -- READ GLOSSARY SETUP

                                Abort if carriage return. Set main utility
                                pointer to glossary start at main memory
                                $0800. Read the glossary character, aborting
                                if an $00 end-of-glossary marker. Scan
                                each character beyond a carriage return for
                                a match with the glossary character. Abort
                                if no match is found. If a match is found,
                                increment the glossary nest pointer by two.
                                Sound alarm if nest depth exceeds eight
                                levels, and reset to level zero. Set the
                                glossary activity flag. Save the starting
                                address of the glossary string to the
                                glossary nest at the current level.

$2AAA-2AEB -- READ CHARACTER FROM GLOSSARY

                                Zero the apple key stash. Get the glossary
                                nest pointer and use it to get the current
                                glossary character to be output. Force
                                feed a self-modifying read to actually get
                                the glossary character. If a carriage return
                                or a $00 end, then pop the glossary stack and
                                decrement the glossary nest pointer by two,
                                resetting to zero on underflow. If a fake
                                carriage return "]", substitute a real one.
                                Exit with read character or substitution in
                                the accumulator.

$2AFD-2B28 -- FILL GLOSSARY SETUP

                                Clear the screen if not WPL. Set the main
                                utility pointer to $0800, the glossary start.
                                Read the glossary and print it to screen,
                                showing sixteen lines at a time if full
                                screen and eight lines at a time if split.
                                A pressed key gets another 8 or 16 entries,
                                aborting on the $00 end-of-glossary marker.

```

## Listing C.9—cont.

## \$2B29-2B6C -- FILL GLOSSARY

Put down a new definition prompt. Get user response, aborting on a carriage return. Replace the final \$00 marker in the glossary with a carriage return. Enter up to 128 characters from the keybuffer into the glossary. On a glossary overflow, ring the ding dong. POSSIBLE ERROR: \$2B55 should be \$10. As is, the glossary overflow will not trip until the entire program is overwritten! Continuously mark the open end of the glossary with \$00 markers after each and every character entry.

## \$2B6D-2B7E -- RETURN EXIT PROMPT

Save the Y register. Print return-to-exit prompt if not WPL. Restore the Y register.

## \$2B7F-2BAD -- AUXILIARY FUNCTION SELECT

Bypass menu if WPL is active. Unsplit and clear screen if not WPL. Set main utility pointer to the auxiliary functions menu. Print menu to screen, aborting on \$3F end marker. Then print return exit prompt. Get user response, forcing upper case. Convert ASCII A-J into 0-9 numeric, aborting if out of range. Double the pointer and get the selected function address pair. Force the address pair on the stack, and then do an indirect jump using the forced sub return method. NOTE: RTS goes to address PLUS one.

## \$2BCA-2BD0 -- TOGGLE PAGE/POSITION FLAG

Switch the page/position flag \$78 between \$00 (no display) and \$80 (display), or vice versa.

## \$2BD1-2C33 -- SHOW PAGE/POSITION SETUP

Clear bottom of screen. Clear the hex buffer at \$C0-C2. Put LOCURS into the auxiliary utility buffer \$AE-AF, and into the screen start pointer \$88,89. Count the number of printable lines, working back from LOCURS to the beginning of the file. Ignore lines with dot commands in them. Put the total into the hex accumulator at \$C0-C2.

## \$2C34-2C51 -- CALCULATE PAGE/POSITION LINK

Find the number of printed lines per page and then subtract the top margin, the bottom margin, and knock one off for either the top or the bottom line if used. This leaves the number of printable text lines per page. Use the hex division subroutine at \$53F5 to divide the total number of printable lines by the printable lines per page. This sub returns with the page count in \$C0 and the line count in the accumulator.



## Listing C.9—cont.

\$2C52-2C82 -- REPORT PAGE/POSITION TO SCREEN

Print [/] Page prompt to screen. Add one to page count, convert to decimal and print to screen. Print Line prompt to screen. Move line count to hex accumulator, add one, and print to screen. Adding one is needed since people count the first line and the first page as one, rather than zero.

\$2C83-2C8E -- ProDOS GET PREFIX MLI

Get the ProDOS prefix and put it in \$1F00. Exit via ProDOS error processor.

\$28CF-2C96 -- CREATE PRODOS SUBDIRECTORY

Get and hold new pathname. Then jump to ProDOS create routine at \$30F6.

\$2C97-2D15 -- QUIT APPLEWRITER

Code needed to return control to another ProDOS application or ".SYS" file. Unsplit the screen. Clear the screen if not WPL. Prompt user. Get user response. Force upper case. If not "Y", then abort. If there was no slot 3 connection before booting, trash the autostart byte and exit immediately. If there was a slot 3 connection before booting, get the connection and put it back in the ProDOS drivers list. Reset ProDOS default values. Init slot 3 connection. Trash autostart byte and exit via ProDOS Quit MLI.

\$2D1C-2D44 -- KEYBOARD TO PRINTER/MODEM SETUP

Clear the screen if not WPL. Set the modem activity flag at \$2E71 to \$40 for modem active. Clear the REFQ flag at \$2E70. Reset the REFQ prompt to all normal text. Put an escape into the active command hold at \$2E72. Prompt the modem setup screen. Grab the printer or modem hooks.

\$2D45-2D95 -- KEYBOARD TO PRINTER/MODEM PROCESSOR

Get character and save to X register. If a \$00 NULL, ignore and get another. If modem is not receiving, process character for possible modem mode change if [esc] or REFQ. If a change, reprompt REFQ and get another character. If not, test (E)cho flag. If in echo mode, print to screen. Test (R)ecord flag and print to file if needed. Then print the character to printer or modem. Finally, get another character, exiting on an [esc]-Q quit. If the modem is already receiving, test [F]ilter flag. If filter is active, ignore all control commands except carriage return, backspace, and delete. Print character to screen. If [R]ecord is active, print character to file as well. Finally, get another character, exiting on an [esc]-Q quit.

## Listing C.9—cont.

## \$2D96-2D9E -- RECORD MODEM TO FILE

If the [R]ecord flag is set, enter the character to the text file. Abort if not.

## \$2D9F-2DB0 -- MODEM PROMPT SETUP

Save the current HPOS and VPOS to the stack. Print the modem prompt to screen in the upper left hand corner. Restore HPOS and VPOS. Bash the vertical position.

## \$2DB1-2DE4 -- MODEM SCREEN PROMPT

Set VPOS and HPOS to the upper left. Bash new VPOS. If modem command hold is an [esc] then print entire prompt to top of screen. If modem command hold is a R,E,F, or Q then print the command in brackets, followed by the (R)ecord .... string. Decrement the window top to prevent scrolling.

## \$2DE5-2E36 -- PROCESS MODEM MODES

If the previous key was not an [esc] then reset the REFQ flag and exit with a cleared carry. If an "R" then change the "R" in the modem prompt string to or from inverse. If an "E" or a "F", likewise change the "E" or "F" to or from inverse. Exit with a set carry if R,F, or Q. Note that the text string serves both as a stash and a hold for the "R", "E", and "F" mode flags. Fall through to next module if not REF.

## \$2E37-2E6F -- QUIT MODEM CLEANUP

If not a "Q" for quit, then abort with a cleared carry. Pop stack twice to cancel subroutine. Reset window to top of screen. If in record mode, reformat screen margins. Prompt for slave mode, saving \$00 to modem activity flag if inactive, and \$C0 if slave.

## \$2E74-2EC6 -- SAVE GLOSSARY TO DISK

Get the pathname, aborting on a carriage return. Set the main utility pointer to \$0800 main memory, the glossary start. Count the active characters in the glossary, terminating on the first \$00 end marker. Calculate the length and force feed the length into the ProDOS Write MLI file for this module. Save the old pathname to the = hold. Move the new path name into the pathname buffer. Set prefix. Set MLI reference number as a text file. Set EOF. Fall through to ProDOS Write MLI, saving from \$0800 to the force fed length in main memory. Go through ProDOS error processor as a subroutine, then restore the = path name and close the glossary file.



## Listing C.9—cont.

## \$2EC7-2ED6 -- LOAD GLOSSARY FROM DISK

Get the pathname, aborting on a carriage return. Jump to the WPL program loader and use it to load the glossary by diverting the load buffer to main memory \$0800.

## \$2ED7-2EDE -- GET AND HOLD PATHNAME LINK

Get pathname via subroutine, aborting on a carriage return. Save the pathname to the pathname buffer at \$1F00.

## \$2EDF-2F0A -- GET FILENAME OR VOLUME NAME

Reset old filename flag \$B0. If not WPL, prompt user for filename. If local \$02 flag is set, prompt user for volume name instead, overwriting the previous prompt. get user string. if a "?", then do a catalog and try again. Otherwise, quit with filename or volume name in the keybuffer.

## \$2F0B-2F35 -- SET MLI REFERENCE FILE NUMBER

Enter at \$2F0B for a textfile, at \$2F13 for a binary file, or at \$2F1B for a binary file that needs created first. Open the file. Get the reference number and move it to the tab write, tab read, prt write, prt read, main read, and main write MLI buffers.

## \$2F36-2F59 -- MOVE PATHNAME TO PATHNAME BUFFER

On a carriage return, exit subroutine if old filename flag not set. If set, exit both subroutine and calling code by popping the stack. Otherwise, move the keybuffer to the pathname buffer, shoving everything one to the right to make room for a length count. Quit on a carriage return, and then stuff the length count into \$1F00.

## \$2F5A-2F83 -- SAVE TAB FILE

Get and hold the filename. Add a ".TAB" trailer to the filename. Create and open binary file. Save via ProDOS Write MLI, saving the 128 bytes starting at \$1D80. Exit via ProDOS error processor.

## \$2F84-2FB1 -- LOAD TAB FILE

Get and hold the filename. Add a ".TAB" trailer to the filename. Open binary file. Load via ProDOS Read MLI, placing 128 bytes into \$1D80-1DFF. Exit via ProDOS error processor.

## \$2FB1-2FDA -- SAVE PRINT FILE

Get and hold the filename. Add a ".PRT" trailer to the filename. Create and open binary file. Save via ProDOS Write MLI, saving the \$0170 bytes starting at \$B780. Exit via ProDOS error processor.

## Listing C.9—cont.

## \$2FDB-3004 -- LOAD PRINT FILE

Get and hold the filename. Add a ".PRT" trailer to the filename. Open binary file. Load via ProDOS Read MLI, loading \$0170 bytes into \$B780-B8F0. Exit via ProDOS error processor.

## \$3005-301C -- GET "SYS.PRT" AND "SYS.TAB" FILES

Enters the initial print and tab values on bootup. Set SYS filename. Load as tab file. Then load as print file. Then adjust screen margins to new margin values.

## \$301D-3062 -- INITIALIZE A VOLUME

Abort if WPL or glossary is active. Load the formatter code, destructively overwriting the glossary and WPL files. Execute the formatter code starting at \$0800 as a sub. Empty the destroyed glossary and WPL files by zeroing their first bytes.

## \$3063-30C3 -- CREATE AND OPEN A FILE

Add prefix to pathname. Create a text file with reference #2. If file does not exist, open it, and then process any errors. Move the reference number to the MLI routines for close, get EOF, set EOR, and single byte PDB write. If file does exist and matches "=" filename, then proceed as above. If no match existing file, prompt for destroy of existing filename. Get user response. Force upper case. If "Y", then proceed as above. If not, then close all files and re-enter main word processing loop.

## \$30C4-30DF -- PRINT FILENAME TO SCREEN

Save registers. Print filename to screen if not WPL, bypassing the length count, forcing upper case, and quitting on a length count match. Restore registers.

## \$30E0-30F5 -- OPEN FILE TWO

Place an \$02 into the reference number hold. Process slot and drive if needed. Open the file via a ProDOS Open MLI, using \$1F00 as the pathname, and \$BB00 as the file buffer.

## \$30F6-310D -- CREATE SUBDIRECTORY

Process slot and drive if needed. Create a directory using a ProDOS Create MLI, using the pathname at \$1F00, allowing reading, writing, rename, or destroy, and forming a linked subdirectory. Exit via ProDOS error processor.

## Listing C.9—cont.

## \$310E-3125 -- CREATE TEXT FILE

Process slot and drive if needed. Create a text file using a ProDOS Create MLI, using the pathname at \$1F00 that may be modified and is a standard seedling file. Exit via ProDOS error processor.

## \$3126-314D -- ProDOS ERROR PROCESSOR SETUP

Abort if no error. Hold the error number to local stash \$00. Save the WPL and glossary activity status to the stack. Process error message as live screen entry. Restore WPL and glossary status. Return to main word processing service loop.

## \$314E-31AE -- ProDOS ERROR PROCESSOR

Clear the old pathname. Print to screen only. If this was an attempt at running a STARTUP program, then reset the startup flag and abort. If not, set the string activity flag \$AD, close the present file, close all files, clear the screen bottom, and ring the ding dong. Set local pointer \$02,03 to error message file. Print error message leader. Scan the error message file for the error number in low ASCII. If found, print the high ASCII (normal text) error message up to the next error number. If the error number is not on the list, then print the error number directly to the screen.

## \$31A9-31AE -- INCREMENT ERROR MESSAGE POINTER

Advance the local error message pointer \$02,03 by one count.

## \$31AF-31D4 -- READ ONE CHARACTER FROM MEMORY

Increment the auxiliary utility pointer. Test to see if a switch from LOFILE to HIFILE is needed. If so, make the switch. Read the next character from the text file and force low ASCII. If not a delete, return with the read character intact. If a delete, return with a \$00 in the accumulator instead.

## \$31D5-31E9 -- READ FROM MEMORY SETUP

Update LOCURS pointer. Hold initial cursor position in the main utility pointer. Set the auxiliary utility pointer to the start of LOFILE.

## \$31EA-3200 -- READ FROM MEMORY OR DISK SETUP

Test the copy from memory flag \$77. Go to previous module if a read from memory. If a read from disk, set up one sector read of disk. Force feed buffer of \$B900-\$BAFF.

## Listing C.9—cont.

\$320B-324E -- READ ONE CHARACTER FROM SECTOR BUFFER

If the disk sector buffer \$B900-BAFF is empty or has been completely read, read a new 512 character sector to this buffer. Read the character pointed to by \$BD and put it into the accumulator. A forced dynamic address change is used in \$3242 to allow access of 512 bytes without needing double wide pointers. Close file if final character read is a \$00 end marker.

\$324F-3264 -- ProDOS READ MLI

Read 512 characters from disk into the \$B900 buffer. If end of file, exit without processing error. If any other error, exit via ProDOS error processor.

\$3265-328C -- TAB SETUP

Calculate the present POS from the last carriage return. Clear and prompt screen bottom if not WPL. Get user response. Force upper case. Process tab Set, Clear, or Purge. Update the tab status display.

\$328D-32A9 -- CLEAR ONE TAB

Scan the 64 pairs of tab addresses in the tab file, searching for a match to POS. If a match is found, then zero that address.

\$32AA-32B4 -- CLEAR ALL TABS

Set all of the tab addresses in the tab file \$1D80-1DFF to \$00.

\$32B5-32E7 -- SET ONE TAB

Scan through the tab address file to see if the tab is already set. If so, abort. If not, scan through tab address file \$1D80-1DFF to see if there are any available \$00 tabs. If none are available, sound the ding dong and exit. If a \$00 pair is available place the POS tab address value into it.

\$32E8-3355 -- DO ACTUAL [I] TAB

Calculate present POS cursor position. Scan through tab list, finding out if a tab to the right is possible. If a possible tab is found, continue scanning, replacing each possible tab with the lowest possible tab to the right. If not possible, exit by reformatting screen margins. If tab is possible, test open apple key for skip-over. If not skip over, add spaces until LOCURS equals the tab value. If skip over, move characters from HICURS to LOCURS until LOCURS equals the tab value. Abort on end of HICURS \$FF marker. Exit by reformatting the screen margins.



## Listing C.9—cont.

\$3358-33A9 -- UPDATE TAB STATUS IMAGE

Fill the tab image \$B600-B6FF with all dots. Write a 1, 2, 3 ... etc to each tenth position, using the 6502 in DECIMAL (!) mode. Write a suitable "fives" marker to each five slot, using the single quote for 5-95, the exclamation point for 105-195, and the vertical bar for 205-235. Mark the set tabs by scanning through the tab address file, and complementing the character in the tab image for each set tab.

\$33AA-33B2 -- INIT POINTERS AND FLAGS

Init LOFILE. Init HIFILE. Adjust screen margins. Fall through to next module.

\$33B3-33C5 -- INIT FLAGS

Set for normal data line. Use wraparound. Update entire screen; no bottom prompt. Do not display page/position. Set data direction to "<". Reset case flag. Reset verbatium flag.

\$33C6-344E -- MAIN WORD PROCESSING ENTRY

Reset stack. Init flags. Reset busy flag. Turn screen on. Update HICURS pointer. Update LOCURS pointer. Reformat screen margins. Unsplit screen. If startup flag is set, then get and run startup WPL routine, and update print values. Either way, fall through to next module.

\$33F1-344E -- MAIN WORD PROCESSING SERVICE LOOP

Set \$FD flag to load to memory. Set \$77 flag to not copy from memory. Clear load string flag \$79 to allow user input. Clear \$71 to use main memory as disk source. If type-ahead buffer is not empty, and if not the slave modem mode, then update the screen. If WPL is not active, and if the string flag \$AD is not set, then reset the string flag, get user response, and redo the loop. Read the keyboard. If an [esc] and if WPL is active and if not in a WPL subroutine, then quit WPL and repeat the main service loop. Get user character. Get help if open-apple ? or /. Reject all NULL \$00 characters. If a delete key, do the deletion. Otherwise, process character and repeat loop.

\$344F-348D -- FILTER CHARACTER

If WPL but not glossary, process character as if it were a control command. Bypass control testing if [V]. Force low ASCII and process as control command if control command. Reset case flag. Process carriage return as control command. Reset the page/position display flag \$78. Send the character to both file and screen. Reformat screen markers.

## Listing C.9—cont.

## \$348E-34C3 -- FILTER CONTROL COMMAND

Force control code and hold locally to \$C2. Test reformat flag \$CD and reformat screen if set. If an [esc] key, toggle the data display and exit. Turn replace flag \$F5 off unless [R]. Zero mystery flag \$ED. Set main utility pointer to start of control command prompt file. Scan the control prompt file, going one past each "[" seeking a match to the current control character. If a match is found, remember twice its position in the list to the X register. If no match is found, quit on end \$00 marker. Use the X register position pointer to pick an address pair. Shove this pair on the stack, and do an RTS, using the forced sub return method to do an indirect jump to the intended control function. Note that the jump goes to the forced address PLUS ONE.

## \$34F9-3528 -- SPLIT SCREEN SETUP

If WPL is not active, clear the bottom of the screen and put down the [Y] user prompt. Get user response and clear prompt. Force upper case. If a "Y", turn split screens on. If a "N", turn split screens off. If a carriage return, swap screens only if [Y] is already active.

## \$3529-3533 -- SWITCH TO OTHER SPLIT SCREEN

Abort if [Y] is not active. Switch screens by changing the "V" slot of \$F8 from one to zero, or vice versa. Jump to screen pointer fixer.

## \$3534-353F -- TURN SPLIT SCREEN ON

Force [Y] flag to split screen on, lower screen active. Set the split screen pointer \$F9,FA to the present LOCURS position. Jump to screen pointer fixer.

## \$3543-354E -- TURN SPLIT SCREEN OFF

Abort if WPL is active. Reset split screen flag to \$00 for one full screen. Jump to recalculate the vertical screen position.

## \$354F-357D -- FIX SPLIT SCREEN POINTERS

Update the screen. Exit if split screen is off. Check which split screen is active, and calculate VPOS position, using \$00 for the top screen and \$0C for the bottom screen, twelve lines down. Save the static cursor position to \$98,99. Save the present LOCURS cursor position to \$F9,FA. Move characters from HIFILE to LOFILE or vice versa as needed to get to the desired point in the file. Init LOFILE on underflow.



## Listing C.9—cont.

## \$357E-3593 -- CALCULATE SCREEN WINDOW

If upper split screen, set window top at \$00 and window bottom at \$0C. If lower split screen, use \$0C and \$18. If full screen use \$00 and \$18. Set horizontal cursor to left. BASH the vertical screen address.

## \$3594-359E -- TOGGLE DATA LINE DISPLAY

Advance the \$E5 data display flag to its next of three possible values, with \$00 being no display, \$80 being the usual MEM-LEN-POS display, and \$C0 being the tab display.

## \$359F-35AD -- TOGGLE WRAPAROUND MODE

Abort with ding dong if the screen right margin is not set to 80 characters, since broken words are only allowed on a full screen display. If a full screen display, change the \$E1 wraparound flag either from or to its \$00 whole words, or its \$FF broken words setting.

## \$35AF-35B5 -- TOGGLE CARRIAGE RETURN DISPLAY

Change the \$74 carriage return flag either from or to its \$00 normal display or \$FF show returns as an inverse "M" character.

## \$35B6-35BC -- TOGGLE VERBATIUM FLAG

Change the \$72 [V]erbatium flag either from or to its \$00 normal use of characters or its \$FF imbed control characters directly into text state.

## \$35BD-35C2 -- PROMPT SCREEN BOTTOM AND GET RESPONSE

Short code link to first prompt the screen bottom and then get the user response.

## \$35C3-35D8 -- FORMAT FILENAME AND PRINT TO SCREEN

Clear and prompt screen bottom. Print the old filename to the screen. Get any user changes to the filename. Abort on a "?" for catalog or a carriage return. If the "=" filename is to be used, scan the old filename to the first comma, and then copy the keybuffer beyond the comma. If a new filename, transfer the entire filename. Zero out the rest of the filename buffer. Hold the filename length to \$E9.

## \$360B-3623 -- CLEAR AND PROMPT SCREEN BOTTOM

Abort if WPL is active. Clear bottom of screen. Print the selected control command prompt to screen, stopping on the first space. Then print a ":" to screen.

## Listing C.9—cont.

\$3624-3637 -- SET CURSOR FOR BOTTOM WINDOW

Set HPOS to zero. Set VPOS to 9 lines from window top if split screen, or 21 lines from window top if full screen. Makes room for three lines, usually a blank line, a prompt, and a second blank line. BASH VPOS.

\$3638-3646 -- CLEAR BOTTOM OF SCREEN

Set cursor for window bottom. Clear window if not WPL. Increment and BASH VPOS.

\$3648-366B -- SAVE TEXTFILE SETUP

Zero save/adjust flag \$3647. Use auxiliary memory as file source by setting \$71. Save the old filename to the "=" buffer. Print old filename to user prompt. Get any user changes to the filename. If a "?", then catalog disk, restore old filename, and try again. Abort on a carriage return. On any other filename, fall through to next module.

\$366C-36C9 -- SAVE ENTIRE TEXTFILE TO DISK

Set \$AD flag to use old filename. Read the last character in the filename. If "+" then set the Append flag \$E2 and zero the "+" out of the filename. If not, clear \$E2. Copy LOCURS to \$98,99, remembering present cursor location when finished saving. If adjust bypass processing that follows. If save, filter filename for special delimiters. If any are found, process via next module. Move cursor to end, putting everything in LOFILE. Save file to disk. Update screen markers. Reset old string flag and adjust flag. Move enough characters from LOFILE to HIFILE to restore old cursor position.

\$36CA-3726 -- SAVE PART OF TEXTFILE TO DISK.

Process special delimiters. Zero last delimiter. Save present cursor position to auxiliary utility pointer. This transfers to the actual disk write routines as a starting address. Begin moving characters from HIFILE to LOFILE, one at a time. Compare each character against the first character in the delimiter string. If no match, keep scanning characters. If a match try to match next character in the delimiter string. Substitute carriage returns and bypass wildcards as needed. If no perfect match, abort by moving all the characters back to the original LOCURS position, and then restoring the old filename. If a match is found, write to disk, using \$AE,AF to mark the start of stuff to be saved, and \$84,85 LOCURS to mark the end. Then update the screen markers, reset the string flag \$AD, move everything back to the original LOCURS position, and restore the old filename.

## Listing C.9—cont.

\$3727-3747 -- SET FILENAME COUNT

Move the filename at \$B700 into the filename hold at \$1F00, carefully shifting everything one to the right to make room for the length count. Count only numerals, letters, commas, and periods, stopping on any control command or any other punctuation. Save the length count to \$1F00.

\$3784-377C -- WRITE TEXTFILE TO DISK SETUP

Force entire file to low ASCII. Abort if only an adjust, rather than a save. Find length by subtracting LOCURS from the aux utility pointer \$AE,AF. Open filename. If append, set mark via MLI. If not append, set end of file via MLI. Save as many sectors as needed to disk using next module. Close file when finished.

\$377D-37DC -- WRITE ONE SECTOR TO DISK

Read first character of the portion of the text file to be saved into \$B900. Continue copying until no more characters needed or until buffer fills at 512 characters. On each buffer filling, write a new sector to the disk. Read aux or main memory per flag \$71, using aux memory for a normal textfile save. Save length to data buffer. Exit via ProDOS error processor.

\$37DD-37F8 -- CLOSE FILES

Enter at \$37DD to close all files. Enter at \$37E5 to close any one file. Poke file reference number to MLI buffer, using \$00 for all files, and the number on any one file. Close the files with a ProDOS Close MLI. Reset buffer to close all.

\$37F9-380B -- ProDOS GET EOF MLI

Find the length of the currently open file, reporting to the buffer at \$380C. Load the registers with the possible 24 bit length result, with X = MSB (normally zero) Y = middle 8 bits, and A = LSB.

\$3811-3830 -- ProDOS SET MARK MLI

Copy the reference number from set mark to get EOF. Save new end-of-file marker from X register MSB (usually zero), Y register middle eight bits, and Accumulator LSB. Do set mark MLI, setting the new endpoint to the file. Used during Append.

\$3831-387C -- POSTFIX SLOT AND DRIVE SETUP

Abort if the first character in the filename is not a comma or period. Zero the slot and drive stashes at \$38D3, 38D4. Read the filename. Force upper case. If a slash is found, hold distance to slash in \$38D4, and

## Listing C.9—cont.

fall through to next module. If nothing beyond comma or period, also fall through to next module. If a D for drive, insert 0 for drive 1 or 1 for drive 2 into MSB of \$38D3. If a S for slot, force range to 0-7 and shift to put \$38D3 into ProDOS DSSS 0000 format.

## \$387D-38D5 -- POSTFIX SLOT AND DRIVE

Check the DSSS 0000 stash. If slot zero, force slot six instead. Get volume name from ProDOS On-line MLI using this slot and drive. Truncate to sixteen characters maximum, loading into \$B900 path name buffer. If a postfix exists, append it onto the path name. Add a slash to the start of the pathname, and Then move the pathname to the \$1F00 buffer. Count the characters in the pathname and save the character count to \$1F00.

## \$38D6-38E5 -- FIND PATHNAME OF INTENDED SLOT AND DRIVE

Store the DSSS 0000 in the accumulator into the MLI buffer. Do ProDOS On-line MLI to load name of disk in target drive into path name buffer at \$B900. Exit via ProDOS error processor.

## \$38E6-38FD -- OPEN TEXT FILE

Calculate filename length and save to \$1F00. Use \$04 text file and reference number \$01. Open file via ProDOS Open MLI. Transfer reference number other MLI links. Get EOF of current file and save to EOF stash at \$3918 (low), \$3919 (med), and \$391A (high).

## \$391B-3928 -- ProDOS SET EOF MLI

Reads the current open file and appends the current end position. Position must have been pre-poked into \$3926 (low), \$3927 (med), and \$3928 (high). Exit via ProDOS error processor.

## \$3929-3949 -- PROCESS SPECIAL DELIMITERS

If the usual "/" delimiter that does not allow any fancy stuff, put an \$01 into \$EC, the any length stash, an \$02 into \$EB the fake carriage return stash, and an \$03 into the \$EA wildcard stash. If a special delimiter is used, put the next higher ASCII character into \$EC, the next one after that into \$EB, and the next one after that into \$EA. For instance, a "<" delimiter will have an any length "=", a fake carriage return ">", and a wildcard "?". Note that these four ASCII characters follow each other in sequential numeric order.

## \$394A-398D -- LOAD SETUP

Hold the current LOCURS position in \$BA, BB formatting pointer. Set the old string flag \$AD to \$FF. Set the memory load flag \$71 to



## Listing C.9—cont.

```

$FF for a load into auxiliary memory. Save
the old filename to the "=" file. Force
normal case, load rather than append, and
included delimiters. Print filename to
user prompt and get any changes, saving new
filename length to $E9 stash. Read the
last filename character and compare it to
UT, usually a "\" for screen-only load. If
a screen only load, zero the symbol and set
the screen-only flag $FD. If a question
mark as the first character, do a catalog and
try again. If a carriage return, fall thru
to cleanup. If a legal filename, fall thru
to next module.

```

## \$398E-3A0E -- LOAD PROCESSING

```

Zero out the three delimiter stashes at
$A2-A4. If the first character is a "#" for
copy from memory, then set the copy from
memory flag $77. Scan the filename for
delimiters. If delimiters found, then
process special delimiters, and zero the
final delimiter out of the filename. If
delimiters exist, scan the filename and save
the delimiter positions to $A2-A4. If
delimiters exist, check the final character
for "A" or "N". If "A", set all-occurrence
flag $D2. If "N", set exclude delimiters
flag. Set up disk or memory read. If no
delimiters, read directly to textfile from
textfile or disk. If delimiters, fall thru
to next module.

```

## \$3A0F-3AAB -- LOAD WITH DELIMITERS

```

Get first character from $B900 buffer and
hold to line justify buffer. Read load
string. If a wildcard, get next character.
If a fake carriage return, substitute a
real carriage return. Compare for a match.
If no match, get next character. If a match
and if delimiters to be included, then
continue matching and put first delimiter
into memory. Continue reading characters,
watching for the second delimiter if present.
If third delimiter is present and delimiters
are to be omitted, swallow end delimiter
string from textfile to swallow buffer.
If no third delimiter, read characters till
end of file. If all occurrence flag is
set, repeat search for new first delimiter
string as often as needed. Fall through
to load cleanup module below.

```

## \$3AAB-3ACA -- LOAD CLEANUP

```

Close file. If load was to screen only, put
down return prompt. Restore old filename if
it exists in the "=" file. Reformat screen
margins.

```

## Listing C.9—cont.

## \$3ACB-3B52 -- FIND SETUP

Clear the case flag \$C4, the all occurrence flag \$E2, and the page/position flag \$78. Set the continue search flag \$AD. Clear and prompt the screen bottom. Print the old search string to the prompt, and get any user changes. Abort on carriage return. If "=" then use old find string by moving it to the keybuffer. Save the new find string to the "=" buffer, thus updating to the latest find. Get the first delimiter and process special delimiters. Zero out the search pointer \$A2 and the replace pointer \$A3. Scan the string for delimiters, saving them to pointers \$A2 and \$A3. Check for an "A" at the end and set the all occurrence flag \$E2 if found. Fall through to next module.

## \$3B53-3BF9 -- SEARCH TEXT FOR FIND MATCH

Put LOCURS into the search pointer \$98-99. Read the keyboard and exit on [esc]. Scan for match, setting the any length flag \$C9 if needed. If a wildcard, skip the match for that character. If a fake carriage return, substitute a real \$0D carriage return. Compare the match character against the cursed character in the textfile. If a match, then continue trying for a match on successive characters. If no match, continue the search, going in the direction set by the data direction flag \$CF.

## \$3BFA-3C21 -- FIND FOUND PROCESSOR

Reprompt screen bottom and get user response. If a "Y", do the replacement. Reprompt for continue after replacement. Force upper case and abort if not a carriage return.

## \$3C22-3C4E -- REPROMPT FIND SCREEN

Update the screen. If not WPL, clear screen bottom and reprompt. If replace, add the replace trailer as well. Get user response. Force equivalent control command, and return.

## \$3C51-3C74 -- DO REPLACEMENT

Move the "found" string from HIFILE into a work buffer at \$1C00, aborting on the \$FF high end of HIFILE, and repeating if all occurrence. Then move the replacement string into LOFILE, using real carriage returns, and substituting characters as needed from the \$1C00 file for wildcards and any length strings. If the data direction is ">" then move characters from HIFILE to LOFILE to get to the start of the string for a resumed search. For either data direction, reformat the screen markers and exit, preserving accumulator to stack.



## Listing C.9—cont.

\$3CAC-3CCA -- MOVE CHARACTER BETWEEN LOFILE AND HIFILE

Subtract the \$99 search pointer from LOCURS. If less than LOCURS, then move characters from LOCURS to HICURS. If equal to LOCURS, then abort. If greater than LOCURS, then move characters from HICURS to LOCURS. Abort if you get to the start of LOFILE or the end of HIFILE.

\$3CCD-3CF6 -- NEW PROMPTER

If WPL is not active, then clear the bottom of the screen and put down the erase verify message. Get user response and force upper case. Abort if not "Y". Init pointers and flags. Erase old filename.

\$3CF7-3CFD -- FORCE UPPER CASE

If a lower case character, subtract \$20 to force upper case. NOTE: This is a sloppy case forcer as certain punctuation will also get changed.

\$3CFE-3D0C -- OUT OF MEMORY CLEANUP

If WPL is active, then quit WPL. Set the old string flag \$AD. Ring the ding dong. return with a \$FF in the accumulator.

\$3D0D-3D1D -- UPDATE CASE CHANGE FLAG

Check the case flag \$E8. If the first entry then force upper case to \$C0 flag. If a repeat entry, then change from upper to lower or vice versa.

\$3D1E-3D24 -- TOGGLE DATA DIRECTION FLAG

Change the data direction flag \$CF from or to \$00 "<" or \$FF ">".

\$3D25-3D2B -- TOGGLE REPLACE FLAG

Toggle the replace mode flag \$F5 from or to \$00 (no replacement) or \$FF (replacement).

\$3D2C-3D54 -- CALCULATE POSITION IN PARAGRAPH

Save LOCURS position to \$AE auxiliary utility pointer. Back \$AE up to the first character in the present paragraph. Subtract this position from LOCURS, holding the difference in TAB position pointer pair \$96,97.

\$3D55-3E57 -- UPDATE STATUS LINE

Check display flag \$E5. If normal display, calculate POS, and then put down header in inverse, using high ASCII tab values, and fixing the mouse nest when needed. Home the cursor. Display "<", ">", "U", or "L" in inverse to the first character of either the normal or tab status display. If a tab

## Listing C.9—cont.

display, show the image of the tab file at \$B600 to the top screen line in inverse. Then advance the top of the display window one count downscreen, and BASH this address while exiting. If a normal display, test and display "V" or "R" mode flags. If the type ahead buffer is busy, show a "\*" to the next slot. If wraparound (whole word breaks) is in use, put an inverse "Z" in the next slot at the top of the window. Calculate the remaining memory by subtracting HIMEM-LOMEM, converting the result to decimal and printing it in inverse. Calculate position by subtracting the starting address \$0800 from LOFILE. Display position in inverse decimal. Verify 80 column screen. Display TAB value in inverse decimal. Get filename from \$B700 stash and display in inverse, aborting on 28 characters maximum. Print inverse spaces to end of line.

## \$3E58-3E68 -- PRINT INVERSE DECIMAL TO SCREEN

For up to five digits, get the decimal digit from the decimal accumulator. If a zero, substitute an inverse space for all leading zeros. Fix mouse nest and print to screen in inverse.

## \$3E69-3E7C -- FIX MOUSE NEST

If an inverse ASCII character is coded from \$60-7F, subtract \$40 from it so it will display as an inverse upper case character, rather than as a mouse nest character.

## \$3E7D-3E84 -- GET CHARACTER FOR PRINTING

Read the character in LOFILE that the printer pointer is pointing to. Force low ASCII. Return with character in accumulator.

## \$3E85-3EAA -- FORCE FILE TO LOW ASCII

The text file holds characters in low ASCII except for the LAST character in each screen line, which is set to high ASCII. Save the aux utility pointer \$AE. Read LOFILE, forcing all characters to low ASCII and erasing any old end-of-screen-line high ASCII markers.

## \$3EB2-3EDA -- UPDATE SCREEN MARKERS

Enter at \$3EB2 if from LOCURS, or 3EBA if from aux utility pointer \$AE, AF. Pass UT to screen line formatter. This allows breaks on an underline token, as well as on a whole word space. If present, the UT appears as the last line character. Clear reformat needed flag \$CD. Back up two screen marks and reformat two lines if not to beginning of file. Only the last two screen lines normally need reformatting.

## Listing C.9—cont.

\$3EDB-3F79 -- MARK ONE SCREEN LINE

Start at the beginning of the screen line and begin scanning characters, looking for a carriage return, an end of HIFILE marker, or a match to the right screen margin. Allow for a switch to HIFILE, handle the possible end-of-line underline token, and resolve left screen margin offset as you do the scanning. When an end is found, remove all high ASCII characters from the present screen line, and then mark the end of the present screen line in high ASCII.

\$3F7A-3F82 -- SET POINTER TO HICURS

Move a copy of HICURS into the auxiliary utility pointer \$AE,AF.

\$3F83-3FA4 -- CASE CHANGER

Save the most significant bit of the character since this is a screen marker flag. If the case flag is set to "U", then subtract \$20 only from legitimate lower case letters, forcing them to upper case. If the case flag is set to "L", then add \$20 only to legitimate upper case letters, forcing them to lower case. Restore MSB screen marker.

\$3FA5-3FE5 -- UPDATE SCREEN SETUP

If format needed flag \$CD is set, then reformat screen markers. Abort if screen display flag \$F7 is cleared for no display. Calculate left screen margin. Add \$50 or eighty characters to set the right screen margin. Set Bash needed flag \$8B. Set screen VPOS to top of screen window. Test split screen flag and prompt flag, using a 24 line screen if not split, a 12 line if split, and three lines less either way if the page/position display is active. Save this bottom updateable line to \$C8. Bash VPOS. If the status line flag is set, update either the main or tab status line. Fall through to next module.

\$3FE6-4044 -- FORMAT SCREEN LINE TO BUFFER

Read the keyboard, saving any keystrokes to type-ahead buffer. Back pointer up to the end mark on the last screen line. If \$00, switch to HIFILE. Get and hold screen character, moving it to the screen formatting buffer at \$1C00. Check wraparound flag \$E1. If words may be broken, fill buffer with 79 characters. If a carriage return and if cr flag \$74 is set, then save an inverse "M" to the buffer. If words may not be broken, fill to 79 characters, and then back up to previous space. "Erase" the old word fragment by overwriting it with spaces. Upon fallthrough, line is formatted and pointer points to start of first needed whole word for the next line.

## Listing C.9—cont.

\$4045-4081 -- PRINT LINE TO SCREEN

Take the 80 characters in the screen line buffer and route all even characters to the main memory screen store, working from high to low. Then route all odd characters to the auxiliary memory screen store, again working from high to low. Fall through to next module.

\$4082-40B8 -- CONTINUE SCREEN UPDATE

Increment VPOS and Bash it. If additional screen lines are needed, format and print them one at a time. If page/position flag is set, add page/position display to window bottom. Set cursor to LOCURS-HICURS change in middle of window and Bash VPOS. Set cursed character to inverse ASCII.

\$40B9-40C4 -- GETKEY LINK

If WPL is not active, get user response while preserving the Y register contents.

\$40C5-40DC -- SWITCH SCREEN POINTER TO HICURS

Abort if X register is \$00. Remember the HPOS at which the switch takes place to \$8A, so cursor can later be correctly positioned. Set the screen pointer \$88-89 to HICURS.

\$40DD-40F5 -- PUT CHARACTER ON SCREEN IMMEDIATE

If a printable character, force high ASCII for normal text. If a control character, force low ASCII for inverse text. Divide the Y-register horizontal position by two and place an even character to the even screen page or an odd character to the odd screen page, using the Bashed value of VPOS held in screen pointer pair \$28,29.

\$40F6-4158 -- WPL ERROR PROCESSOR

Save error number to \$C2. Clear bottom of screen and quit WPL. Print WPL error prompt. Scan through the WPL error message file and count carriage returns. When the number of carriage returns matches the error number, then print that error message to screen. If error zero for "Label not Found", then print the unfound label to the screen. Put down return prompt and await user response.

\$415A-4179 -- PRINT/PROGRAM REPROMPTER

Re-entrant code is used to allow continuous refresh of P/P values until a carriage return rather than a P/P value is entered. Show the P/P values. Clear the screen bottom. Save prompt pointer and get new response, then restore prompt pointer.



## Listing C.9--cont.

## \$417A-41A5 -- PRINT/PROGRAM PROMPTER

Clear and prompt screen bottom. If a "?" display P/P values and reprompt via above module. Save the P/P command first letter to \$A2 and the second letter to \$A3. Try to process the P/P command, putting a value into the P/P stash, or doing the WPL command. If a carriage return, exit the reprompter by double-popping the stack.

## \$41A6-41B3 -- STRIP P/P COMMAND FROM KEYBUFFER

Remove and destroy the first two characters from the keybuffer by backing everything up by two characters, aborting on a carriage return. Removes the P/P command but saves the "argument", such as a filename or a numeric value.

## \$41B4-4215 -- SUBSTITUTE (X)-(Z)

Scan the characters in the \$1F00 workbuffer, looking for a "(". See if there is a ")" two spaces further on. If so, check inside and force upper case. Range check on "X", "Y", or "Z". If a WPL numeric is found, change it to decimal and put it in the keybuffer. If no WPL numeric, then transfer from \$1F00 back into the keybuffer. Repeat until a carriage return, making as many substitutions as are needed.

## \$4216-4236 -- SCAN FOR [P] MATCH

If WPL is active, substitute (X)-(Z). Convert to hex. Scan the P/P functions list seeking a match to both \$A2 and \$A3. Abort if no match found.

## 44237-4287 -- ENTER NUMERIC P/P VALUES

Hold twice the match value to \$C6. If a WPL command, fall through to next module. Test the sign bit, unless PM, which is always relative. If absolute, add the new P/P value to zero and store it. If relative, add the new P/P value to the old one and store it. Note that subtraction is done by two's complement addition for negative values. If the PN command, move new PN to running page counter \$BE-BF. If any margin command, adjust the screen margins.

## \$4288-42B9 -- PROCESS NON-NUMERIC P/P COMMANDS

If UT, save character to \$B8DE, replacing UT with \$00 if a space or a carriage return. If a justify command, convert command to \$00 = LJ, \$01 = FJ, \$02 = CJ, or \$03 = RJ and save to justify stash \$B8E0. If a WPL command, get address out of WPL command address file, shove on stack, and do an indirect jump by way of the forced subroutine return method. Note: jumps to listed address PLUS ONE.

## Listing C.9—cont.

\$42BA-42E6 -- WPL COMPARE STRING CS

Ignore any leading spaces in the keybuffer. Hold the leading delimiter to local stash \$00 and scan for second delimiter, aborting on a carriage return. Hold first delimited position in Y register, second in the X register. Scan the length of the first string. If all characters match, exit with \$AD cleared. If no match, set \$AD to \$FF.

\$42E7-4304 -- WPL SUBROUTINE SR

If WPL subroutine stack is more than 32 deep, exit and ring the ding dong, putting down an error message. Save the WPL program counter to the WPL subroutine stack and increment the WPL stack pointer by two. Then do a WPL GO as an unconditional jump.

\$4305-431A -- WPL SUBROUTINE RETURN RT

If nothing is on the WPL subroutine stack, exit and ring the ding dong, putting down an error message. Get the current address pair out of the WPL stack and force it on the WPL program counter \$A0,A1. Decrement the WPL stack pointer by two. Program resumes where it left off, before the sub.

\$431D-4320 -- WPL DISPLAY ON/OFF

Enter at \$431D for ND and \$431E for \$YD. Store a \$00 to the display flag to turn the display off. Store a \$FF to the display flag \$F7 to turn it on.

\$4321-4354 -- WPL STRING LOAD LS

Identify which WPL string. Saving LOCURS, load the WPL string from disk or from memory immediately below HICURS in HIFILE. Then move the loaded string into the keybuffer and zero HICURS. Force carriage return onto end of string in keybuffer. Note: string space is "borrowed" from unused area below HICURS. This routine will not work properly if memory is within 64 characters of being full.

\$4357-4385 -- WPL STRING IDENTIFIER

Scan the keybuffer, looking for an "=" that is followed by a "\$". If found, get next character, force upper case and range check for "A" through "D". Convert string letter into offset in \$02 such that "A" = \$00, "B" = \$40, "C" = \$80, and "D" = \$C0. These will then point to the respective string start in the WPL string file.

\$4386-4397 -- SET UP PRINT TO DISK PD8

Clear the screen if not WPL. Prompt and get the PD8 filename. Handle source and drive if changed. Create and open textfile.



## Listing C.9—cont.

\$439A-43AB -- WRITE ONE CHARACTER TO DISK

Used by PD8. Enter with character in the accumulator and save to MLI buffer. Do a ProDOS Write MLI of a single byte, using the saved character as a data value.

\$4397-43C0 -- SET PRINTER, THEN STALL

Set the printer destination with the next module. Then delay for half a second.

\$43C1-4414 -- SET PRINTER DESTINATION

Reset modem activity flag. Get the print destination. Multiply by 16 and save to \$DE stash. If a PD8, set up print to disk, then force continuous paging. Then set print destination pointer to \$4397. If PD1-PD7, save the old character output hooks. Load the character output hooks with \$C100 if slot one, \$C200 if two, etc. Output a NULL to activate the printer or modem card. Save new printer hook to \$9E,9F. Note that this gives the printer card a way to modify its \$C100 warm link to \$C103 or whatever. Restore old printer hooks. Then send baud rate and other commands to printer or modem card. If PD0, set print to screen hooks by setting printer destination to \$4415.

\$4415-441C -- PRINT TO SCREEN LINK

Abort on a linefeed. Otherwise print to screen via main routine at \$2413.

\$441D-44A0 -- SHOW PRINT/PROGRAM VALUES TO SCREEN

Unsplit the screen. Clear the screen if not WPL. Set the main utility pointer to point at the print/program background copy. Print one numeric entry at a time, down through CR. After each prompt, get the needed print/program value from the \$B8C0 stash, convert it to decimal, and place it in the right slot. Follow with a carriage return. Print the underline token directly to the screen. Bypass the (X)-(Z) values in the print/program stash when getting numeric values. Print justify prompt and convert the justify mode to ASCII LJ, FJ, CJ, or RJ, and print to screen. Print the top line prompt, then the top line. Ditto for the bottom line. Print a return prompt if not WPL.

\$44A1-44BA -- PRINT SINGLE PRINT/PROGRAM PROMPT

Enter at \$44A1 to start with a carriage return, \$44A6 otherwise. Print the selected prompt from the prompt file up to its ending \$3D marker, aborting on the \$00 end of file. Print a single space after the prompt.

## Listing C.9—cont.

\$44BB-44C7 -- SAVE OLD FILENAME TO = BUFFER

Move the current filename in the \$B700 buffer to the filename hold "=" buffer at \$1F40. Hold the most recent character in the X register. Abort on a \$00 ending marker.

\$44C8-44D4 -- RESTORE OLD FILENAME FROM = BUFFER

Move the held old filename in the \$1F40 "=" buffer to the current filename buffer \$B700, aborting on a \$00 end marker.

\$44D5-44E5 -- LOAD GLOSSARY VIA WPL DO ROUTINE

Turn glossary off. Set accumulator to \$08, the starting page of the glossary load. Go to the next module, tricking the module to load a textfile into the glossary, rather than into the WPL file.

\$44E6-453B -- WPL DO MODULE

Set the WPL activity flag \$DF to MSB on. Bypass initing DO routine if a glossary load instead. If really WPL, set the WPL program counter to \$1000, the start of the WPL file. Reset the subroutine flag \$E7, and the old string flag \$AD. Save the old filename to the "=" file. Move the filename of the WPL routine into the current filename buffer. Do a disk read, loading either into the glossary or WPL file as requested. If an overflow, sound the ding dong and put down an error message. If no overflow, when finished, mark the end of the glossary or the WPL file with a carriage return and an ending \$00 marker.

\$453C-456A -- TRANSFER KEYBUFF --> FILE NAME BUFFER

If the old filename is to be reused because of an "=" command, set the \$B0 filename flag and exit. If a new filename, clear the \$B0 filename flag. Bypass all leading spaces in the filename. Move the filename from the keybuffer \$0200 to the filename buffer \$B700 starting with \$B701 and ending on a carriage return. Save the length count to \$B701. Fill the rest of the filename buffer with all zeros.

\$456B-4580 -- NOT REFERENCED

This code module also moves a copy of the keybuffer into the filename buffer, except there is no length count and no zeroing of the rest of the filename. Upper case is also forced. Apparently no longer used.

\$4581-45AF -- WPL INTERPRETER

Check keyboard for [esc] hit and quit WPL if present. If string flag \$AD is active, get

## Listing C.9—cont.

string character from \$A-\$D file. Clear apple key stash. Test for end of command line carriage return. If not, read next WPL character per the next module. If a carriage return, check next character and exit if end of WPL. If a label is present, skip over all non-space characters in the label. This leaves the WPL program counter pointing to the command control character in the next WPL line.

## \$45B0-45D2 -- TEST WPL FOR STRING

Get the character from the WPL program file. Quit WPL if \$00. Increment the WPL program counter and hold the character to \$8C. If the previous character was an "=" (held in X Register from last trip), and this one is a "\$", then initialize a string read with the next module.

## \$45D3-45EF -- SET UP WPL STRING

Convert the A-D into an index \$00, 40, 80, or \$C0. Increment the WPL program counter. Read the first character. If \$00, then no string is present, so zero the \$F6 string flag and go get another character. If a real character, advance the string pointer by one and exit with the first character in the accumulator.

## \$45F0-4603 -- QUIT WPL

Reset the WPL activity flag \$DF, preserving the glossary activity status. Reset the key strobe. Turn the display on. Equalize the keybuffer pointers. Exit with \$FF in the accumulator, which cancels the final WPL interpreter loop.

## \$4604-4607 -- ENTER BOTTOM LINE BL

Trick the next module into handling the bottom line by giving it a 128 character offset so it points to the bottom line file rather than the top line file.

## \$4608-4627 -- ENTER TOP LINE TL

Move the keybuffer into the \$B7C0 top line buffer or the \$B840 bottom line buffer, ending on a carriage return. Ring the ding dong if more than 128 characters in string. Mark the end of TL or BL with a \$00 marker.

## \$4628-4635 -- PAD BOTTOM LINES

Add space and carriage return to each bottom line until the last printable line \$7C matches the page interval \$B8CE.

## Listing C.9—cont.

## \$4636-4643 -- PROCESS CONDITIONAL FORMFEED

Test conditional formfeed \$C0. If not, do unconditional formfeed with next module. Subtract the line counter from the last line hold to find the remaining room on the page. Compare room on page against conditional feed \$C0. If formfeed is needed, fall thru to next module.

## \$4644-4652 -- DO UNCONTIDIONAL FORMFEED

Print a space and carriage return to each line until the running line counter matches the last line stash. Set the end of page \$FF flag and exit.

## \$4653-465C -- PRINT SPACE AND CARRIAGE RETURN

Print a space and a carriage return as needed to pad blank lines on printout. WARNING: introduces a possible "page creep" bug to top and bottom lines. TL and BL printing routines should enter this at \$4658 instead so that these lines are no longer than the rest of the printed lines. Creep problem usually shows up when both printer and Applewriter are set to RM80, and the TL or BL is suppressed on the first page.

## \$465D-46AC -- WPL UNCONDITIONAL JUMP

Set WPL flag \$DF without hurting glossary status. Read the keybuffer, advancing beyond any spaces to get to the start of the label. Hold label pointer to local stash \$05. Reset subroutine flag \$E7. Reset WPL program counter to start of WPL file at \$1000. Scan through the WPL program with the WPL program counter, stoping one beyond each carriage return. Search label for an exact match, exiting on a space or a control command. If no match, go on to one beyond the next carriage return. do this until a match is found or the entire WPL program is scanned. If no match, exit with LABEL NOT FOUND error message and print the label not found. Hold the first label character to \$8C and exit with the WPL program counter pointing just beyond the label on the found line.

## \$46AD-46D5 -- WPL PRINT TO SCREEN

Scan the keybuffer, printing one character at a time to the screen. Abort on either a carriage return or on the "=\$" needed by the PIN command.

## \$46D6-4713 -- WPL GET USER INPUT

Print the PIN line up to "=\$", using the previous module. Find the string A-D character and range check. Set string pointer offset so that A = \$00, B = \$40, C = \$80, or D = \$C0. Hold offset to local



## Listing C.9—cont.

stash \$02. Save the WPL and glossary flag status \$DF. Get string from user. Restore \$DF status. Test for string assignment needed. Fall through to next module if the string needs assigned.

## \$4714-474C -- WPL ASSIGN STRING

If a new string assignment, use the WPL string identifier to calculate the string offset and verify legality. Move the string from the \$0200 keybuffer into the string file \$1E00, using an offset of A = \$00, B = \$40, C = \$80, and D = \$C0. Continue until a carriage return, and end with a forced \$00 marker.

## \$474D-4778 -- SETUP TL OR BL PRINTING

For bottom line BL use an offset of \$80 and the \$B840 file. For top line TL use an offset of \$00 and the \$B7C0 file. Calculate the RM-LM printable line length. Save the old underline mode, and start without any underline. Zero local left, center, right routing flag \$00. Fill the line formatting buffer \$1F00 with all spaces. Mark the end of the formatted line with a \$00. Save the first TL or BL delimiter to \$E6. If \$00, then bypass next module and do not print a TL or BL.

## \$4779-47C8 -- FORMAT TOP OR BOTTOM LINE

The top or bottom line is read from its compact form (delimiters and "#") buffer at \$B7C0 or \$B840 and is then expanded (spaces and full page number) into the formatting buffer at \$1F00. The \$0200 keybuffer is used as a temporary work area, to simplify entry of page numbers and the center- and right-justification process. This keybuffer works in a "batch" mode, in which it first formats the left justified portion and moves it to the line buffer, followed by center, and finally right. Should a "#" crop up, the running page number is substituted into the keybuffer. Pointer \$00 keeps track of the three trips needed, with \$00 = left, \$01 = centered, and \$02 = right. The left string enters at the left margin. The center string starts half the number of center characters shy of center. The right string starts the number of right characters shy of extreme right.

## \$47C9-47F3 -- PRINT FORMATTED TOP OR BOTTOM LINE

If a "\" is used as a delimiter and if the running page number equals the first page number, then restore old underline mode and exit. This suppresses TL and BL on the first page. If a following page or a different delimiter, then pad the left margin, followed by the TL or BL, followed by a space and a carriage return. Note that the extra space

## Listing C.9—cont.

can cause page creep if both printer and word processor are set to 80 columns.

\$47F4-480D -- GET DECIMAL PAGE COUNT

Used to substitute for "#". Save registers. Get running page number \$BE,BF and convert to left justified decimal ASCII. Move result to keybuffer where the "#" would have been. Restore registers.

\$480E-4817 -- ENABLE OR DISABLE PRINTING

If EP0, Store a \$00 to printing flag \$B8, disabling printing. If EP1 or any other non-zero value, store a \$FF to printing flag and allow printing.

\$4818-4822 -- INIT PAGE NUMBER

Move the starting page number in \$B8CA,B8CB into the running page counter \$Be,BF.

\$4823-4831 -- PAD TOP MARGIN

If top margin value in \$B8C6 is zero, abort. If not print space and carriage return for each line needed for the top margin.

\$4832-4843 -- PAD BOTTOM MARGIN

If a short page or a conditional formfeed left a shy bottom margin, add one space and one carriage return per printed line, done as often as needed to make the running line counter \$7C match the last printed line \$7D. Then get the bottom margin value from \$B8C0, and fall "up" to the previous module to pad the remaining lines on the page.

\$4844-4860 -- NEW PRINTING SETUP

Move the PN page number into the running page counter. Set printing pointers. Hold LM and RM to temporary stashes \$5E31 and \$5E32. Turn underline off. Enable printing. Jump to page printing routine.

\$461F-4864 -- CONTINUE PRINTING SETUP

Jump to appropriate point in the page printing routine. This allows printing to pick up where it left off, preserving position on the page.

\$4867-488B -- SET PRINTING POINTERS

Set the first line in paragraph flag \$7F. Reset the key strobe. Grab printer hooks, then stall. Hold LOCURS into the \$98,99 pointer so cursor can be returned to present position after printing. Do a [E], moving everything into LOFILE. Set printer pointer \$90,91 to start of LOFILE at \$0801. Clear end-of-printing flag \$76.



## Listing C.9—cont.

## \$488C-48E0 -- PRINT ONE PAGE

Add one to running page counter. If single page, prompt for return. Zero the running line counter \$7C, the footnote line counter \$FE, and the done with page text flag \$FF. Get the number of printed lines and subtract the bottom margin, saving to the last printed line hold \$7D. Decrement by one if there is to be a bottom line. Print the top line, followed by the top margin padding. Print text body. When enough text lines are printed, handle footnotes. Print the bottom margin padding. Print the bottom line if used. Print enough spaces and carriage returns to bring PL up to PI. If not done, repeat for another page. If to the end of the print file, or if the end-of-printed lines flag \$FF is set, fall through to next module.

## \$48E1-4901 -- DONE PRINTING CLEANUP

If PD0 and WPL is not active, then prompt user for carriage return. If PD8, print a final NULL \$00. Restore screen cursor to original position. Close all files.

## \$4902-4912 -- PRINT TEXT BODY

Abort if the running line counter \$7C equals the final printer line stash \$7D. If not, print one line. Keep repeating until out of text, a conditional formfeed, or out of room on the page.

## \$4913-4920 -- ADJUST PARAGRAPH MARGIN

If the first line in the paragraph, add the relative PM to the absolute LM and hold to adjusted left margin flag \$B7.

## \$4921-4948 -- SETUP ONE LINE PRINT

Read keyboard. If [esc], then set the end of printing flag \$76, and quit WPL if WPL is active. Save the point at which the printing ceased, and move the cursor to this location by moving HIFILE to LOFILE or vice versa as often as needed. If no escape, adjust the line start for the paragraph margin. Format one line, aborting if \$FF flag is set. Justify the line. Print the formatted line.

## \$4949-4952 -- END OF PRINT FILE CHECK

Read next character in LOFILE. If end \$00, set the end flag \$76 and exit with a carriage return in the accumulator.

## \$4953-4987 -- UNDERLINE DETECTOR

Calculate remaining space on line. Get next character from file. If the UT underline token, then toggle flag \$E0. If at end of

## Listing C.9—cont.

line, swallow all spaces until the next printing character.

## \$4988-49AC -- FOOTNOTE DETECTOR

Save the present horizontal line position. Read the next file character, setting end of file flag \$76 if the \$00 at the high end of LOFILE, and forcing a carriage return. Test for a "(", followed by a "<". If the start of a valid footnote, save footnote and continue beyond the footnote. Resume with the first character beyond the footnote next to the last character before the footnote.

## \$49AD-49C6 -- IMBEDDED COMMAND DETECTOR

Ignore if not a period or if the last file character. If a period and the very first character on the line, then process the .XX selected print command. Note that all characters following the .XX go to the key buffer for print/program use, and are not normally printed.

## \$49C7-49F0 -- SAVE FILTERED CHARACTER TO LINE BUFFER

At this point, any remaining characters are real and not part of an underline command, a footnote, or an imbedded command. Save the character to the line formatting buffer \$1C00. If a carriage return, hold the short line count to \$7E, and set the first line in paragraph flag \$7F. If an underline token, adjust line length. If a space and the first character on the line, swallow space. Advance the printer counter \$90,91. Note that the first line flag is also the "don't justify" flag. See Listing C.7.

## \$49F1-4124 -- PRINT FOOTNOTES

Save old underline status. Begin footnote with no underline. Bypass if no footnotes. Set local footnote buffer pointer \$00, 01 to \$1400, the start of the footnote file. Print space and carriage return. Pad left margin with spaces. Get the current footnote character, exiting on a \$00. Print the character. Advance the footnote pointer and continue printing until all footnote lines are printed. Restore underline mode.

## \$4A25-4AAB -- SAVE FOOTNOTE LINE

If the first footnote for this page, zero the footnote buffer and set the footnote line counter. Then, knock two counts off the number of printable body lines. If a second or higher footnote for this page, decrement the footnote count and knock only one count off the number of printable body lines. Begin moving characters from the text file into the keybuffer \$0200, watching for the ending "<" delimiter. Sound the ding dong and exit with an error message if more than

## Listing C.9—cont.

128 characters per footnote line. Terminate keybuffer entry with a carriage return and a \$00 end marker. Advance printer pointer by three to bypass footnote stop marker. Scan the footnote buffer \$1400 to find the open \$00 end marker, sounding ding dong on a main footnote buffer overflow. Move the footnote line in the keybuffer into the open end of the footnote file, sounding ding dong on overflow. On any line or buffer overflow, quit WPL. Move cursor to beginning. Grab screen hooks, close file and exit with error #4 FOOTNOTE OVERFLOW in the accumulator.

\$4AAC-4AD7 -- PROCESS IMBEDDED PRINT COMMAND

Get the first character of the printing command. If end of file, set the stop printing flag \$76. Otherwise, move the entire line starting with the imbedded command into the keybuffer at \$0200. Save registers. Do imbedded command. Restore registers.

\$4AD8-4AE0 -- CALCULATE SPACE LEFT ON LINE

Subtract the current line position \$B7 from the right margin stash \$B8C4 and save as room remaining stash \$75.

\$4AE1-4AEA -- CALCULATE AVAILABLE LINE LENGTH

Subtract the right margin hold \$5E32 from the left margin hold \$5E31 and save as room remaining stash \$75. Note that these saved values cause all top and bottom lines to have constant widths, regardless of margin changes imbedded in the text.

\$4AEB-4B1B -- PRINT FORMATTED LINE

Pad the left margin. Print the line in the \$1F00 line formatting buffer one character at a time, until an \$00 end marker or a carriage return or until line width \$7E is matched. Print one carriage return. If more spaces between lines are needed, print more carriage returns.

\$4B1C-4B2F -- PAD LEFT MARGIN

Enter at \$4B1C to use original left margin, or at \$4B22 to use current left margin. Print as many spaces as needed to get from extreme left to the desired margin setting.

\$4B30-4B39 -- PICK JUSTIFY MODE

If left justify, do nothing. If a fill justify, branch to the fill justify module. If center or right, fall through to the next module.

## Listing C.9—cont.

## \$4B3A-4B6F -- CENTER OR RIGHT JUSTIFY

Save justify mode to the Y register with \$02 for right and \$03 for center. Subtract the number of characters in the line from the line width. If center justify, divide by two to split the difference. Move the characters to the line formatting buffer \$1F00 either at extreme right or center, starting at the highest character and working backwards. Then, add spaces to every location before the formatted string. Note that spaces are not needed beyond the string, since a carriage return stops the printing.

## \$4B70-4BD3 -- FILL JUSTIFY

The fill justify mode works by adding extra spaces as needed to each existing space, and always starting at the left, thus "expanding" the line to fit the available space. Reset the padding counter \$C7. Abort if the flag \$FF is set. Note: the "first paragraph line" flag doubles as a "don't justify the last short paragraph line" flag, since \$FF gets reset earlier on a carriage return. Count the number of non-space characters in the line. Abort if no padding needed, or if last line in paragraph. Add one space per space, starting at the right by moving the text one slot to the right in the \$1F00 line buffer. If an underline, then adjust for the underline mode. Repeat adding a space per space until line is fully justified. Stop when the line is stretched wall to wall.

## \$4BD4-4BE1 -- ADVANCE PRINTER POINTER TO NEXT LINE

Add the length of the characters actually used during the last printed line to the printer pointer \$90,91. This sets the pointer to the start of the next line, bypassing everything printed or used as footnotes, wrapped words, or imbedded commands.

## \$4BE1-4C34 -- PREPARE ONE CHARACTER FOR PRINTING

Save registers. If an underline token, toggle the underline flag \$E0 and substitute a space for the token. If a carriage return, increment the line counter \$7C. Substitute the NULL character via subroutine. If in underline mode, and if the character is not a space, print an underline, a backspace, and the character. Otherwise, print the character. Restore registers.

## \$4C36-4C40 -- SUBSTITUTE NULL CHARACTER

If a \$1F ASCII user separator [\_] is found, substitute a \$00 NULL character. Note: to change to a substitute NULL sit-in other than \$1F, poke the character to \$4C37. A \$00 value will prevent all NULLs. The \$1F



## Listing C.9—cont.

```

character causes problems with KMI commands
on daisywheels, and creates hassles on some
modems and interface cards.

$4C41-4C5F -- PRINT ONE CHARACTER SETUP

Set up slow serial print. If a carriage
return, get the required carriage return
delay. If modem is active, delay for needed
carriage return time coded into $B8DC.

$4C60-4C77 -- SERIAL PRINTING SETUP

Print the character. If a carriage return,
add a linefeed if the printer delay value
in $B8DC is an odd number.

$4C78-4CA3 -- PRINT ONE CHARACTER

If modem is not active, send the character
to the printing hooks and return. If modem
is active, send character to printing hooks
and then get the serial baud rate from $C09B
(slot 1) or $C0AB (slot 2). Stall as long
as needed to time out baud rate.

$4CA8-4CB8 -- STALL FOR CARRIAGE RETURN DELAY

Enter with the carriage return delay value
divided by two in accumulator, except if
zero, use one. Delay that many milliseconds
such that 0 or 1 = 40 milliseconds, 2 = 80
milliseconds, 3 = 120 milliseconds, etc.

$4CB9-4CF5 -- SHOW HEX PRINT VALUE TO SCREEN AS DECIMAL

Treat PM always as relative number. Test
sign bit for relative or absolute. If both
relative and minus, Print a minus to the
screen and do a 2's complement by first
complementing and then adding one. Convert
to decimal. Print left justified value to
screen if WPL is not active.

$4CDE-4CF3 -- PRINT TO SCREEN LINK SAVING REGISTERS

Abort if WPL flag $DF is set. Save
registers. Print to screen. Restore all
registers.

$4CF6-4CFF -- PRINT TO SCREEN LINK DESTROYING REGISTERS

Abort if WPL flag $DF is set. Enter at
$4CF6 to clear the screen, at $4CF8 to
print a character. Print the character to
the screen.

$4D00-4D6D -- GET STRING

Affirm flashing cursor symbol. Get key.
If a NULL, ignore and get the next character.
if a backspace, backspace if not already at
extreme left. If a right arrow, get the
character already on the screen as it is
copied over. If screen character is in the

```

## Listing C.9—cont.

mouse next, fix it. If a delete, backspace. Store the character to the \$0200 keybuffer, aborting on a carriage return. If more than 128 characters, sound ding dong and force character 128. Replace the end carriage return with a space so screen does not scroll. Print to screen. Mark the end of the keybuffer with a carriage return. This routine normally used at the screen bottom for processing filenames, find strings, etc.

## \$4D6E-4DA4 -- ProDOS ACCESS SETUP

Unsplit the screen. Clear the screen if not WPL. Print the [O] menu to screen. Get user response. Force upper case. Change ASCII A-J to numeric 0-9 and range check. If a valid selection, get address of ProDOS routine selected, and bounce off the stack, using the forced subroutine return method of doing an indirect jump. Note: goes to the pushed address PLUS ONE.

## \$4DB0-4DCD -- SET PRINTER/MODEM INTERFACE SETUP

Fill the keybuffer with 128 spaces. Put Down drive prompt. Get user response. Change ASCII 1 drive number to numeric 0, or ASCII 2 to numeric 1, and range check. Abort if out of range. If valid drive, fall through to next module.

## \$4DCE-4E8A -- SET PRINTER/MODEM INTERFACE

Move the encoded interface values from the proper drive into work stashes \$82 and \$83, such that \$82 is coded as \$PPP0 0000 and \$83 is coded \$SDDD BBBB, where S = stop bits, D = data bits, B = baud rate, and P = parity. Note: this is the standard command form as needed by the 6551 serial interface chips. Put down format prompt and print current parameters to screen. Get user response. Scan the input baud rate against the baud rate list for a match. If no match, exit with a ding dong and an INVALID PARAMETER error message. If a valid \$0-F baud rate, store to low half of \$83. Affirm a one in the fifth slot. Read the user response and get the data bits needed. Range check for 5 through 8 bits. Encode the data bits so that 5 = 11, 6 = 01, 7 = 10, and 8 = 00. Save the data bits to \$83, while preserving the other bits. Get the parity character and compare against S for space, M for mark, E for even, O for odd, or N for none. Encode so that S = 111, M = 101, E = 011, O = 001 or N = 110. Save as high three bits in \$82 local stash. Get the stop bits and range check on 1 or 2, and convert to a 0 or 1 numeric. Save as MSB in \$82. Update interface values in:

\$B8E2 - \$PPP0 0000, slot 1  
 \$B8E3 - \$SDD1 BBBB, slot 1  
 \$B8E4 - \$PPP0 0000, slot 2  
 \$B8E5 - \$SDD1 BBBB, slot 2



## Listing C.9—cont.

## \$4E8B-4EE4 -- PRINT INTERFACE DATA TO SCREEN

Abort if WPL is active. Get startup bit info from \$82, coded PPP0 1011. See the IIC Reference Manual for more info. If the first attempt to set interface, change to default 0,8,N,1 prompt. Mask out the baud rate bits from \$83, coded \$SDD1 BBBB. Get equivalent baud rate from table as 16 bit hex value and convert to decimal. Print to screen as left justified decimal ASCII. Get the data bits and decode into 5, 6, 7, or 8. Display as ASCII numeric. Get the parity bits from \$82 and use a table to convert to ASCII E, M, N, O, or S, then display to prompt. Print a comma. Get stop bits and display to screen.

## \$4EE5-4EF2 -- PRINT HEX PAIR AS DECIMAL

Enter with the low byte in the accumulator and the high byte in the X register. Save to hex buffer. Force a 16 bit conversion. Print as left justified decimal ASCII. Print an ending comma.

## \$4F15-4F56 -- SEND INTERFACE VALUES TO SERIAL INTERFACE

Verify Apple serial interface in legal slot. Abort if some other brand card is present. Get the print destination and verify slot one or two. If a baud rate of zero, bypass and use default values. Store \$PPP0 0000 to \$C09A for slot one or \$C0AA for slot two. Store \$SDD1 BBBB to \$C09B for slot one or \$C0AB for slot two. Check for an "old" or "new" machine. Make the following pokes to the I/O ram slots:

\$0579: FF if slot 1 and "new" (cr = 255)  
 \$057A: FF if slot 2 and "new" (cr = 255)  
 \$06F9: FF if slot 1 and "old" (cr = 255)  
 \$06FA: FF if slot 2 and "old" (cr = 255)

\$06F9: 00 if slot 1 and "new" (no video)  
 \$06FA: 00 if slot 2 and "new" (no video)  
 \$07F9: 00 if slot 1 and "old" (no video)  
 \$07FA: 00 if slot 2 and "old" (no video)

The \$FF values set any port-forced carriage returns to the highest possible value, while the \$00 commands defeat any video echo. More on this in the IIC Technical Reference Manual.

## \$4F58-4F6F -- VERIFY APPLE SERIAL INTERFACE CARD

Get the print destination and force feed an address read. Verify \$C10B as \$01 and \$C10C as \$31 if slot one. Verify \$C20B as \$01 and \$C20C as \$31 if slot two. Exit with Z flag set if a legal Apple interface exists. Matches either the internal IIC ports or a Super Serial card in a IIC.

## Listing C.9—cont.

## \$4F70-4F9F -- ADJUST MARGIN SETUP

Clear bottom of screen. Bypass prompt if WPL is active. If not, prompt screen bottom with adjust prompt and get user response. Force upper case. Abort if not "Y". If "Y", set adjust flag \$3647, which changes the [S]ave routine so it adjusts only. Clear bottom of screen. Put down one moment prompt. Adjust margins using [S]ave.

## \$4FA0-4FD6 -- SET SCREEN FORMAT

Enter at \$4FBA for unconditional set, or at \$4FA0 to set only if both LOFILE and HIFILE are empty. Find the difference between the left and right margin. If this difference exceeds 240 characters, substitute 240 as maximum value to screen right margin hold \$B1. Force the whole word break mode by setting [Z] flag \$E1.

## \$4FD7-5024 -- LIST ProDOS VOLUMES ON LINE

Clear the screen if no WPL activity. Print volumes on line header. Get entire list of volumes on line via ProDOS On-line MLI, loading into \$B900. Get the coded slot and drive from the buffer and print slot and drive to screen. Then print a space and a "/" slash. Print the volume name from the \$B900 file. Add a 16 character offset to the pathname and repeat for each volume on line. The \$B900 buffer has sixteen slots for each volume name. These all start with a byte encoded \$DSSS 0000 encoding the slot and drive, followed by a slashless volume name, and ending with a \$00 marker. A \$00 first byte signifies the end of the list.

## \$5027-504C -- DECODE AND SHOW SLOT AND DRIVE TO SCREEN

Enter with \$DSSS 000 in the accumulator. Print the word "slot" followed by a space to the screen. Downshift to \$0000 DSSSS and then mask 0-7 for \$0000 OSSSS. Change to ASCII and print the slot number 0-7. Move the drive bit in the carry and add one, so that \$01 = drive 1 and \$02 = drive 2. Convert to ASCII and print on the screen. Note sneaky use of re-entrant code.

## \$504D-5065 -- PRINT STRING TO SCREEN

Enter with the string starting address high in the accumulator and low in the X Register. Force feed address into load command. Get the first character of the string and print to screen if not WPL. Increment the force fed address. Repeat printing characters until a \$00 ending marker.

## Listing C.9—cont.

## \$5066-50AB -- RENAME ProDOS DISK FILE

Prompt for present name and get response.  
 Handle slot and drive and save present name  
 to \$1F40 buffer. Prompt for new name and  
 get response, holding in \$1F00 buffer.  
 If a "?", then do catalog and try again.  
 If valid new name, then do ProDOS Rename MLI  
 using \$1F00 as the new name buffer and \$1F40  
 as the old name buffer.

## \$50AA-50FD -- LOCK OR UNLOCK ProDOS DISK FILE

Enter at \$50AA to unlock or at \$50AE to  
 lock. Get filename from user. Get file  
 attributes from ProDOS, loading into \$50E2  
 buffer. If file is to be locked, put a \$01  
 into \$50E5 forcing the file to read only.  
 If file is to be unlocked, put a \$C3 into  
 \$50E5, allowing the file to be read, written,  
 renamed, or destroyed. Return the file  
 attributes to disk by doing a ProDOS Set  
 Attributes MLI. Exit via error processor.

## \$50FD-510F -- DELETE A ProDOS DISK FILE

Get and hold filename. Adjust for slot and  
 drive. Delete file via a ProDOS Delete MLI,  
 using filename buffer \$1F00. Exit via the  
 ProDOS error processor module.

## \$5124-5174 -- CATALOG SETUP

Reset catalog flag \$5123 to screen only.  
 Get and hold volume name to \$1F00. If a  
 "#" as the first character, set catalog flag  
 \$5123 to text file. Then knock one count off  
 of file length. If a carriage return as  
 the only character, use old volume name. If  
 a new name, get the prefix and hold to \$1F00.  
 Handle slot and drive if needed. Get the  
 attributes for the selected volume using a  
 ProDOS Get Attributes MLI. Move attributes  
 to work buffer at \$5157 so that \$5157-58 is  
 the auxiliary data, \$5159 is the type of  
 file, and \$515A-5B are the blocks in use.  
 Open the volume directory using a ProDOS  
 open MLI, holding the catalog at \$B900.

## \$5175-51AE -- PRINT CATALOG HEADER

If a catalog to screen, clear screen if not  
 WPL. Print the volume name to the screen.  
 Print an opening parenthesis followed by  
 the date and time, followed by a closing  
 parenthesis. Print a "V" for version. Get  
 and print version number. Print column names  
 to screen using "Type ... Blocks ...", etc.

## \$51AF-5231 -- PRINT CATALOG BODY

Print up to 15 catalog entries if full screen  
 or only 7 entries if split screen. Print a  
 carriage return. Get the file descriptive  
 entry for the next file in the list. Fall  
 through to cleanup if first character in file

## Listing C.9—cont.

is not a "/". Check the file descriptive entry to see if file has been deleted. If deleted, get next entry. Check to see if file is locked. If so, display a "\*" in the first screen slot. If not, print a space. Get the file type and print to screen. Add spaces to tab to eighth column. Read number of blocks and print to screen as decimal ASCII. Add spaces to tab to fifteenth screen column. Print the file name. Add spaces to tab to thirty-first screen column. Print the created time and date. Add spaces to tab to forty-seventh column. Print the modified time and date. Add spaces to tab to sixty second column. Get file length as 24 bit word, convert to decimal, and print. Repeat until screen is full or out of files.

## \$5232-52A7 -- PRINT CATALOG FOOTER

If more files remain, and if catalog to screen, wait for user to hit a key. If [esc] then print footer and clean up. If any other key, and print a new screen of file entries. When finished, Close the directory and print a carriage return. If file is a directory, calculate blocks used and print availability message. If catalog to screen, put down return prompt and get user response. If catalog to file, clear the catalog to file flag and update the screen markers.

## \$52A8-52DF -- PRINT DECIMAL VALUE TO SCREEN

Enter at \$52AC for one hex digit, at \$52A8 for two digits, at \$52C5 for four, or at \$52B9 for five digits. Convert to decimal. Print to screen, supressing leading zeros and replacing them with spaces.

## \$52E0-52EE -- TAB SPACES

Enter with horizontal tab value in the Print spaces from the present screen position to the tab position, "erasing" anything that was on the screen.

## \$52EF-5317 -- PRINT PRODOS FILE TYPE

If a filetype above \$10, substitute \$0C for a SYSTEM file. Multiply the \$0-F file type by six to point to a six letter file tipe in the \$5B5A file. Print six characters to screen as a file type.

## \$5319-534A -- GET FILENAME FOR CATALOG

Filenames in the directory are seperated by \$27 characters. Add \$27 to the present filename pointer to get the start of the next one. If not an overflow, read directory from \$B900 buffer. If an overflow, get another directory page and put it in the \$B900 buffer. Set pointer to \$B904, the first descriptive entry in the buffer.



## Listing C.9—cont.

\$534B-538C -- GET DATE AND TIME FOR CATALOG

Clear access flag \$5398 for year and month, rather than hours and minutes. Read the date from byte pair %YYYY YYMM ~~MM~~ DDDD and print the month, followed by a slash, followed by the day, followed by a slash, followed by the year. Switch access flag to hours and minutes. Read the time from byte pair %000H HHHH 00MM ~~MM~~., and print the hour, followed by a colon, followed by the minutes.

\$538D-5397 -- PRINT SLASH AND DECIMAL VALUE

Print a slash, followed by a two digit hex value converted to decimal ASXCII.

\$5399-53A9 -- PRINT VOLUME NAME OR FILENAME

Hold name lenght to \$53AA. Read filename, printing that many characters to screen or to textfile.

\$53AB-53B8 -- PRINT CATALOG FILENAME HEADING

Get the "Type .... Blocks ...." etc from the \$5BDE stash and print it to screen or textfile, aborting on a \$00 end marker.

\$53B9-53DD -- HEX TO ASCII DECIMAL CONVERTER

Enter at \$53C2 if hex value is already in hex buffer C0-C2; at \$53BC for a full 24 bit conversion holding MSB in the Y register, middle 8 bits in the X register, and LSB in the accumulator; or at \$53AB for a 16 bit conversion holding the LSB in the accumulator and the MSB in the X register. Zero the decimal result buffer \$0380-0385. Do a single digit conversion, change to decimal low ASCII and hold to decimal result buffer. repeat until nothing remains in the hex buffer. LEAST significant decade always ends up in \$0380, regardless of length. The X register holds the number of digits in the result on exit.

\$53DE-53F4 -- SINGLE DIGIT HEX CONVERSION

Set maximum bits to be converted to 24. Multiply the remainder in the hex buffer by two, and transfer any overflow to the accumulator. Keep repeating until you get a result of ten or more in the accumulator. When this happens, subtract ten from the accumulator and increment the low hex byte \$C0. Exit with decimal value in the accumulator and the residue corrected to the correct remainder. Sounds strange and not at all obvious, but it works. Fast and short.

## Listing C.9—cont.

## \$53F5-540B -- DIVIDE LINES BY LINES PER PAGE

Enter with the total number of lines in the hex buffer \$C0-C3, and the lines per page in \$80. Zero accumulator. For 24 trips, multiply hex buffer by two and accumulate the overflow. Every time the accumulator exceeds the lines per page, subtract the lines per page and increment the hex buffer. Exit with the page count in \$C0 and the line count of the last page in accumulator. If you liked the previous module, you will love this one. A very fast and neat divide routine.

## \$540C-546F -- ASCII DECIMAL TO HEX CONVERTER

Clear 16 bits worth of hex buffer \$C0,C1. Set arithmetic mode flag to absolute \$FF if no sign preceeds the decimal number in the \$0200 keybuffer. Set arithmetic mode flag to relative \$2B or \$2D on a "+" or a "-". Swallow any spaces and get first decimal ASCII numeral. Range check for 0-9. If valid, convert the ASCII numeral to a one digit BCD \$0-9 value. Take whatever is already in the hex buffer and multiply it by ten. Do this by doubling the value and holding it to X and Y; then double twice again. Finally add eight times the hex buffer to two times the hex buffer and save the result back to itself. After the 10X multiplication, add the BCD digit to the hex buffer. Repeat the process as long as legal numerals are found in the keybuffer. When finished, check the mode flag. If a negative number, do a 2's complement on the hex buffer by subtracting the buffer from zero. Exit with the signed binary value in \$C0 low and \$C1 high, and the mode flag with its N bit set for absolute or cleared for relative.

## \$5470-5474 -- MULTIPLY HEX BUFFER BY TWO

Multiply contents of hex buffer by two by shifting all the bits one to the right.



**Listing C.10. How to customize Applewriter AWD.SYS.****A. For Personal use only:**

Use the methods of module six, installing your own custom patch and then saving to a new backup diskette.

Note that selling or otherwise passing on copies is an absolute no-no when done this way.

**B. Preferred Commercial method:**

Create a booting program and store it to a ProDOS diskette as the first system file. This program should prompt the user to insert his Applewriter disk and then use the ProDOS MLI interface to load AWD.SYS to \$2000. It should then install the needed patches in the needed locations. Finally, it should do a \$2000G or a CALL 8192 to run the customized program.

This method can be safely sold or given away, so long as the end user provides his own legal copy of AWD.SYS.

**IMPORTANT:** When AWD.SYS is opened with the ProDOS MLI, the file buffer **MUST** be set to \$BB00.

.....

Note that a ProDOS license is needed to sell or give away any software using the ProDOS operating system. The annual costs start at \$50. Contact Apple directly for more details.

**Listing C.11. How to capture AWD.SYS source code.**

**Note:** The capture process is done under DOS 3.3e as much less sand gets kicked in your face this way.

This capture process works **ONLY** on AWD.SYS and **ONLY** by using DISASM IIe. Any changes at all to either program will cause one or more problems.

- (1) Initialize two diskettes under DOS 3.3e using the system master disk. Call these disks one and two. Use the HELLO program of listing 8.3.
- (2) Put a copy of DISASM IIe onto both diskettes. Then add a CONVERTed copy of AWD.SYS to both disks.
- (3) Add GRABBER.D.12 (Listing 8.4), GRABBER.D.34 (Listing 8.5) and SNEAKY.D (Listing 8.6) to both of the diskettes. These programs are also available, ready to CONVERT, on the companion diskette.
- (4) Cold boot the first diskette. Answer "Y" and "1". Wait a long time while strange and wondrous things flash by on the screen and while other things go beep in the night. Two source code textfiles, named AWD.SOURCE1 and AWD.SOURCE2 will eventually get generated and placed on-disk. Be patient!
- (5) Repeat step (4) for the second disk, answering "Y" and "2". Two source code textfiles, named AWD.SOURCE3 and AWD.SOURCE4 will appear on-disk. Figure 8.1 shows how these four textfiles are related to AWD.SYS.
- (6) Verify the four source code modules by assembling them under EDASM and comparing them against the original. All object code must **EXACTLY** agree with AWD.SYS.

Listing C.12. Applesoft HELLO program used to capture AWD.SYS.

```
1000 D$ = CHR$ (4): PRINT D$;"PR#3": TEXT : HOME : CLEAR

1020 PRINT "          ***** AWD.SYS Sourcecode Grabber *****"
1030 PRINT
1040 PRINT "          To capture the AWD.SYS sourcecode, you
1050 PRINT "          first must create TWO new disks, under
1060 PRINT "          DOS 3.3e. Both disks must contain ALL
1062 PRINT "          of these files and NO others:
1070 PRINT
1080 PRINT "                  HELLO (this program)
1090 PRINT "                  AWD.SYS (converted)
1100 PRINT "                  DISASM IIE
1110 PRINT "                  GRABBER.D.12
1120 PRINT "                  GRABBER.D.34
1130 PRINT "                  SNEAKY.D
1150 PRINT
2000 PRINT "          OK to continue (Y/N) ? -----> "; GET ZZ$

2010 PRINT : PRINT
2020 IF ZZ$ = "Y" OR ZZ$ = "y" THEN 2040
2030 END
2040 PRINT "          Which disk is this (1/2) ? --> "; GET ZZ$

2050 PRINT
2060 IF ZZ$ = "1" THEN 3000
2070 IF ZZ$ = "2" THEN 3010
2080 GOTO 1000
3000 PRINT CHR$ (4);: PRINT "EXEC GRABBER.D.12": END
3010 PRINT CHR$ (4);: PRINT "EXEC GRABBER.D.34": END
9999 END
```

## Listing C.13. Details of EXEC file GRABBER.D.12.

```
REM
REM   AWD.SYS Source code grabber
REM
REM   To use, EXEC GRABBER.D.12
REM
REM
TEXT: HOME: CLEAR
PRINT CHR$(27); CHR$(17)
BLOAD DISASM IIE
BLOAD SNEAKY.D, A$2000
POKE 2091,22
POKE 2761,51
POKE 3162,32: POKE 3163,24
POKE 3164,253 :POKE 3165,09
POKE 3166,128
POKE 2473,96
BLOAD AWD.SYS, A$4000
CALL2048
1
4000
5003
2000
40DE
40DF
11
454D
454E
11
458F
45BF
11
4AED
4AFC
11
4C86
4C88
11
4C8C
4C8E
11
4D08
4D0A
11
4D0C
4D12
11
4E70
4E73
11
4EB2
4EB4
11
4EBF
4EC6
11
4F73
4F75
11
4F7C
4F83
11
4F9D
```

## Listing C.13—cont.

```
4F9F'
11
4FA9
4FB0
11
4FCA
4FCC
11
4FD3
4FDA
11
4FF4
4FF6
11
4FFD
5003
11

2000
1

NYAWD.SOURCE1

CALL2048
1
5004
60F5
3004
5048
504A
11
505B
5062
11
50EB
50ED
11
50EF
50F4
11
50F5
50F5
11
50FC
50FF
11
5102
510D
11
5114
5116
11
511A
5125
11
5252
5254
11
525D
5264
11
5647
5647
```

## Listing C.13—cont.

```
11
57CD
57CF
11
57D3
57DA
11
57DB
57DC
11
57EE
57F0
11
57F7
57F8
11
57FC
57FE
11
580C
5811
11
5826
5828
11
582C
5830
11
58D3
58D5
11
58DC
58DE
11
58E2
58E5
11
58F9
58FB
11
591E
5920
11
5924
5928
11
5C4F
5C50
11

2000
1
```

## NYAWD.SOURCE2

.....

This program will only work with DISASM IIe and the ProDOS 2.0 AWD.SYS version of Applewriter.

This program is available ready-to-run on the companion diskette for this volume.



Listing C.14. Details of EXEC file GRABBER.D.34.

```
REM
REM      AWD.SYS Source code grabber
REM
REM      To use, EXEC GRABBER.D.34
REM
REM
TEXT: HOME: CLEAR
PRINT CHR$(27); CHR$(17)
BLOAD DISASM IIE
BLOAD SNEAKY.D, A$2000
POKE 2091, 22
POKE 2761, 51
POKE 3162, 32: POKE 3163, 24
POKE 3164, 253 :POKE 3165, 09
POKE 3166, 128
POKE 2473, 96
BLOAD AWD.SYS, A$4000
CALL 2048
1
60F6
7474
40F6
639D
639F
11
63A3
63AA
11
63AB
63AB
11
6CA4
6CA8
11
6CF4
6CF5
11
6EF5
6F14
11
704C
704C
11
709F
70A1
11
70A5
70A9
11
70CB
70CD
11
70DC
70DE
11
70E2
70FC
11
7106
7108
11
710C
```

## Listing C.14—cont.

```
710E
11
7117
7119
11
7120
7122
11
7123
7123
11
7157
715B
11
7318
7318
11
7398
7398
11
73AA
73AA
11
```

```
2000
1
```

```
NYAWD.SOURCE3
```

```
CALL2048
1
7475
7FFF
5475
7475
7FFF
11
```

```
2000
1
```

```
NYAWD.SOURCE4
```

This program will only work with DISASM IIe and the ProDOS 2.0 AWD.SYS version of Applewriter.

This program is available ready-to-run on the companion diskette for this volume.

Listing C.15. *SNEAKY.D* code needed for *AWDSYS* capture.

```
2000- 27 43 C1 C1 00 00 00 00
2008- 27 44 C1 C1 AB B1 00 00
2010- 27 45 C1 C1 AB B2 00 00
2018- 2A B9 C2 C2 00 00 00 00

2020- 2A C6 C2 C2 AB A4 C4 00
2028- 2A C7 C2 C2 AB A4 C5 00
2030- 32 42 C3 C3 00 00 00 00
2038- 32 43 C3 C3 AB B1 00 00

2040- 32 33 C3 C3 AB B1 00 00
2048- 37 A1 C4 C4 00 00 00 00
2050- 37 A2 C4 C4 AB B1 00 00
2058- 37 A3 C4 C4 AB B2 00 00

2060- 3E FD C5 C5 00 00 00 00
2068- 3F 05 C5 C5 AB B8 00 00
2070- 32 45 C6 C6 00 00 00 00
2078- 32 46 C6 C6 AB B1 00 00

2080- 32 47 C6 C6 AB B2 00 00
2088- 4F 58 C7 C7 00 00 00 00
2090- 4F 65 C7 C7 AB B1 B3 00
2098- 4F 6C C7 C7 AB B2 B0 00

20A0- 50 53 C8 C8 00 00 00 00
20A8- 50 54 C8 C8 AB B1 00 00
20B0- 50 55 C8 C8 AB B2 00 00
20B8- FF FF FF FF FF FF FF FF
```

**Listing C.16.** *WPL.TWO COLUMNS* will automatically format your final hard copy into two or more columns. If desired, each column can be separately fill justified or even microjustified.

```

P
P      *** WPL.TWO COLUMNS ***
P
pnd
ppr[L]
ppr  Two Column Formatter:
ppr  .....
ppr
pin   Filename of pd8 formatted left column ---> =$A
pin   Filename of pd8 formatted right column --> =$B
ppr
pin   Character distance between columns -----> =$C
ppr
ppr
ppr      *** busy - please wait ***
P
p set tabs
b
psx$C
ny
al f//*/
y?
u
psx-1
pgoal
e
tp
ts
ny
p load files
l$A
b
f<><{}><a
e
f<<>%$#><
y?
e
l$B
p move right column
b
psyl
bl ppr formatting line (y)
f<>%$#><
[esc]
psy+1
u
u
u
u
f<><
[esc]
d
x
b
f/{}//
[
pgo cl
pgo dl
cl f/{}//

```

## Listing C.16—cont.

```
y?  
p  
i  
x  
d  
h  
f<><<  
y?  
d  
pgobl  
dl p  cleanup  
b  
f/%$#//  
y?  
pqt
```

Gotchas: Both columns must be previously printed to disk as separate and fully formatted ".pd8" files. These files must hold an exact copy of the final printed image. The right column should have a left margin value of zero.

Paired brackets mean imbedded control characters.

To demo this code, use the SAMPLE LEFT COLUMN and SAMPLE RIGHT COLUMN off the companion diskette, along with a column separation of 39.

This program is available ready-to-run on the companion diskette.

# Index

## A

- [A] command, 145
- Accessories for printers
  - platens, 52
  - silencers, 41-42
  - tractor feeds, 41, 50
- AGLOSS command, 5, 89
- AIOIFIER patch, 104, 211
- Apple keys
  - how to use, 31-32
  - sticking of, 24-25
- Applewriter 2.0 (old)
  - executing of string control characters, 14
  - NULLsand, 4
  - running on a Applewriter IIe, 29
- Applewriter 2.0 (ProDOS).
  - See ProDOS
  - Applewriter 2.0
- Applewriter IIc
  - extended memory and, 23
  - printing problems of, 28-29
  - trashing of, 9
- Applewriter IIe (AWIle)
  - (stock)
  - DOS program of, 23-24
  - erasing programs on boot/master disk, 23

- Applewriter IIe (AWIle)
  - (stock)—cont.
  - executing of string control characters, 14
  - extended memory and, 23
  - loss of help screens, 35
  - NULLs and, 4-5
  - patches and, 93-102
  - slow entry and, 33
  - underlining and superscripting on, 5
  - upgrading of, 112-113
- Applewriter IIe patches
  - CLARIFIER, 208
  - CURSIFIER, 205
  - LINKIFIER, 207
  - NULLIFIER, 203
  - PATCHIFIER, 206
  - RESTORIFIER, 209
  - STRETCHIFIER, 204
- Applewriter IIe programs
  - CLARIFIER, 9, 100-101, 179-181
  - CURSIFIER, 8-9, 12, 15, 98-99
  - LINKIFIER, 100
  - NULLIFIER, 5, 61, 96, 172-173
  - PATCHIFIER, 8-9, 99-100, 177-178
  - RESTORIFIER, 102

- Applewriter IIe programs—cont.
  - STRETCHIFIER, 6, 61, 73-74, 97-98, 174-176
- Applewriter III
  - converting files of, 25
- ASCII control commands, 56
- ASCII high and low, 122
- Assembler, 7
- [@] command, 145
- AUTOLETTER, 34
- AUTO.PD8.WPL program, 78
- AWB.SYS, 115
- AWC.SYS, 115
- AWD.SYS, 115, 165
  - how to capture source code, 319
  - how to customize, 318
  - detailed script of main program, 205-317
  - important entry points, 260-264
- AW.SYSTEM, 115

## B

- [B] command, 145
- Backspace commands
  - [closed apple] key and, 31
  - searching for, 12
- Backup copies
  - how to make, 7



Banking of characters, 97  
 Baud rate, 45  
 Beep, 156  
 Binary files  
   converting to text files, 24  
 BLOAD command, 24, 115  
 Bload Patch, 99  
 BOLD PS  
   WPL.FORMAT, 78  
 BOLD PS printwheel  
   punctuation and, 21  
   rearranging, 75  
 BOOTIFIER patch, 105, 213  
 Booting code  
   custom, 161-162  
 Boot/master disk  
   erasing programs on, 23  
 Brackets in glossaries, 82  
 BSAVE command, 115  
 Buffers  
   deletion, 125  
   glossary, 125  
   key, 125, 141  
   modem, 142  
   swallow buffer, 125  
   type-ahead, 141  
 Bullets  
   printing solid, 34  
 BULLET SHOOTER  
   program, 34, 186

## C

[C] command, 145  
 Cables for extending printers, 45  
 CAMERA READY program, 50-51, 64-65, 78, 188  
 Carriage return  
   command, 146  
   imbedding of  
     in the glossary, 31  
 Case changer and NULL, 5  
 Case flags  
   changing of, 145

Catalog  
   how to set, 138  
   improving catalog  
     display, 11  
     placing in text file, 11  
 Centering text, 28, 77  
 Character entry, 141-142  
 .cj command, 71, 77  
 CLARIFIER (AWIIe) patch, 208  
 CLARIFIER (AWIIe)  
   program, 9, 100-101, 179-181  
 Clear screen, 156  
 [Closed apple] keys  
   function of, 31-32  
 Columns  
   printing of, 15  
 Comment lines in glossaries, 86  
 Comments  
   adding, 13-14  
 Conditional execution and  
   WPL, 155  
 Control commands, 145-149  
 CONVERT program, 11, 112  
 Copying text, 10, 32  
 COUNT  
   internal, 139  
 Counters  
   defined, 131, 132  
 CREEPIER patch, 106, 216  
 CURSIFIER (AWIIe), 8-9, 205, 221  
   how to use, 98-99  
   loading strings and, 15  
   searching or replacing  
     backspace  
     commands, 12  
 CURSIFIER (ProDOS 2.0)  
   program, 108

## D

[D] command, 145  
 Daisywheel(s)  
   BOLD PS, 21, 75

Daisywheel(s)—cont.  
   differences among, 21, 49  
   MAJESTIC PS, 75  
   printing of spokes, 20-21  
   print quality, 13  
   rearranging spokes, 75  
   registration, 15  
   TITAN 10, 75  
   underlining, 13  
   versus dot matrix  
     printers, 49-50  
 Data file, 136  
 Delete commands, 145, 148  
 Delete key  
   alternatives for the, 31  
 Deletion buffer, 125  
 Delimiters  
   loading string and, 15  
   searching for, 142, 145-146  
   used in copying text, 10  
 DGLOSS command, 5, 89, 90  
 Diablo compatible printer, 22  
 Diablo 630 daisywheel, 49  
   glossary tricks for, 62  
   imbedded commands in, 54, 59  
   secondline problem of, 22, 75  
 Diablo 630 formatting  
   glossary with  
   tutorial, 194-195  
 Ding-dong  
   how to obtain, 30  
 DISASM IIe, 165  
 Disassembler program, 165  
 Disk labels, 43  
 Disk mailers, 43  
 Disks, 43  
 Do command, 154  
 DOS access menu, 147  
 DOS program of Applewriter  
   IIe, 23-24  
 Dot matrix printers (DMP)  
   daisywheel versus, 49-50

Dot matrix printers (DMP)—  
 cont.  
 formatting glossary with  
 tutorial, 193-194  
 superscripting on, 6

## E

[E] command, 145  
 EDASM, 7  
 Editing a long file, 25  
 EGLOSS command, 5, 89, 90  
 Entry  
   how to deal with slow, 33  
 Entry points, 135-136  
 .ep (enable printer)  
   command, 4  
 Epson MX80 formatting  
   glossary with  
   tutorial, 196-197  
 Epson printers  
   imbedded commands in,  
     54, 59  
   underlining and  
     superscripting on,  
     5-6  
 Error messages, 130  
 Escape sequences  
   imbedding of, 35  
 Extended memory  
   importance of, 23

## F

[F] command, 145-146  
 File(s)  
   clearing to end of, 25-26  
   converting binary to text,  
     24  
   printing of WPL, 27-28  
   print part of a, 4  
 FILE LISTER program, 27-28,  
   184-185  
 Filenames  
   problems with loading,  
     11  
   punctuation and, 10  
 Flags, defined, 131, 132

FLINSERT program, 35, 187  
 Footers  
   printing of, 27  
 Footnotes, 71, 125  
 FORMAT BOLD PS, 78  
 FORMAT MAJESTIC, 78  
 FORMAT NOFRILLS D630  
   program, 69-78,  
   189-191  
 Formatting  
   automatic, 69-78  
   Diablo 630 glossary  
     with tutorial, 194-195  
   DMP glossary with  
     tutorial, 192-193  
   Epson MX80 glossary  
     with tutorial, 196-197  
   footnotes and, 71  
   Imagewriter glossary  
     with tutorial, 198-199  
   memory needed for,  
     71  
   restrictions with, 71  
   paragraph ends and,  
     76-77  
   rearranging daisywheel  
     spokes, 75  
   reformatting, 143  
   right margins and, 71-72  
   setting body  
     microjustification, 76  
   shadowing titles, 77  
   startup of, 73  
   squashticity and, 73-74  
 STRETCHIFIER program  
   and, 73-74, 174-176  
   tightening vertical  
     spacing, 75-76  
   underlining and, 71,  
     74-75  
 Form letters  
   how to create, 34-35  
 Frontspacing  
   [closed apple] key and, 31  
   [U] command and, 148

## G

[G] command, 146  
 GET.FILE.INFO., 138  
 Global value of page zero,  
   133  
 Glossaries  
   adding comments to,  
     13-14  
   [closed apple] keys and,  
     31  
   comment lines in, 86  
   Diablo 630 and, 62,  
     195-196  
   disallowed keys in, 81  
   DMP and, 193-194  
   Epson MX80 and,  
     197-198  
   help screens and, 87-88  
   how to use, 81-82  
   Imagewriter and,  
     199-200  
   imbedding commands  
     using, 55, 60-63  
   imbedding of a carriage  
     return in, 31  
   imbedding of string  
     control characters, 14  
   length of, 20  
   printing of, 27-28  
   reading of, 146  
   restrictions of, 84-86  
   self prompting/self  
     titling, 19, 86-88  
   use of brackets in, 82  
   WPL commands in,  
     82-83  
 Glossary buffer, 125  
 GLOSSIFIER patch, 106, 215  
 GRABBER.D.12, 165,  
   321-323  
 GRABBER.D.34, 165, 166,  
   324-325  
 Grappler problem defined, 7  
 GRAPPLIFIER patch, 104-  
   105, 212

**H**

[H] command, 146

**Headers**

- adding to mailing lists, 26
- printing double, 27

HELLO, 165

- program, 320

**Help screens**

- glossaries and, 87-88
- how they are lost, 35

HICURS, 140

HIFILE, 121-123, 140

**High work files**

- definition of, 118
- globals page, 127
- listing of, 232-233
- print/program file values, 127
- systems page, 127
- tabs and, 126-127

HIRES dumps, 28, 99, 100, 162-164

Hooks, 139

Hyper [delete] key  
how to stop, 31

**I**

[I] command, 146

IGLOSS command, 5, 89

**Imagewriter formatting**

- glossary with
- tutorial, 198-199

**Imbedded escape commands**

- multicharacter, 6-7
- single letters and, 6

**Imbedding**

- a carriage return in the  
glossary, 31
- dot commands, 12
- escape sequences, 35
- hidden lines in a mailing  
list, 26
- how to imbed  
commands, 55
- print commands, 53-55

**Imbedding—cont.**

- string control characters,  
14
- using the glossary, 55,  
60-63
- using verbatim method  
[V], 14, 55, 56, 59-60
- using WPL, 55, 64

**Insert paragraphs, 148****Insert words**

- command for, 148

**Interface**

- ProDOS [MLI], 117,  
136-138

**Interface**

- RS232, 44-45, 46-48

**Interface circuits**

- imbedded commands for,  
54

Internal files, 118-119, 128,  
234-240

**J**

[J] command, 146

Jumps and WPL, 154-155

**Justification**

- how to set right, 29

**K**

[K] command, 146

**Kerning**

- how to do, 13

Key buffer, 125, 141

KEYIN, 141

**Keystrokes**

- how they are entered,  
141-142

**L**

[L] command, 9, 146

**Labels**

- for disks, 43
- making identical, 9  
and WPL, 153-154

LGLOSS command, 6, 89

LINKIFIER (AWIIe) patch, 207

LINKIFIER (AWIIe) program,  
100

Load command, 146

Loading a string from text, 15

Local value of page zero, 133

LOCURS, 134, 140

LOFILE, 120-123, 140

**Low work files**

- definition of, 118
- deletion buffer, 125
- footnotes and, 125
- glossary buffer, 125
- key buffer, 125
- listing of, 227-231
- memory management  
code and, 124
- swallow buffer, 125
- tabs and, 126

**M**

[M] command, 146

Machine language interface  
[MLI], 117, 136-138

**Mailing list**

- adding common headers  
to a, 26
- imbedding hidden lines  
in a, 26
- printing of hidden lines  
on a, 26

MAJESTIC PS, 75

**Margins**

- setting of, 4, 71-72, 145

**Master/boot disk**

- erasing programs on, 23

Memory and ProDOS, 115,  
140

Memory management code,  
124

Memory maps, 116-118

**Microjustification**

- definition of, 51
- fooling a printer to do,  
63, 151
- how to set, 22, 76



MLI access commands,  
136-137

Modem buffer, 142

Modems  
setting of, 138

Monitor access, 138-139

Mouse nest  
problems caused by, 101

Moving characters, 145, 146,  
148

Moving copy, 10

## N

[N] command, 147

NEC Spinwriters, 49

NULLIFIER (AWIIe) patch,  
203

NULLIFIER (AWIIe)  
program, 5, 61, 96,  
172-173

NULLIFIER (ProDOS 2.0)  
patch, 105-106, 214

NULLs  
case changer and, 5  
how to imbed, 4-5

## O

[O] command, 147

[Open apple] keys  
function of, 31, 63

Overtyping  
how to deal with, 32-33

## P

[P] command, 147

Page creep, 30-31, 76

Page numbers  
printing of, 27

Page/position command, 149

Page zero  
detailed script of use,  
247-259  
how to use, 131-132,  
134-135  
locating, 132-134  
summary of use, 244-246

Paper, 42, 51

Patches for Applewriter IIe  
(AWIIe), 93-102  
CLARIFIER, 208  
CURSIFIER, 205  
how to patch, 94-95, 202  
LINKIFIER, 207  
NULLIFIER, 203  
PATCHIFIER, 206  
RESTORIFIER, 209  
STRETCHIFIER, 204

Patches for ProDOS 2.0,  
102-108

AIOIFIER, 211  
BOOTIFIER, 213  
CREEPIFIER, 216  
CURSIFIER, 221  
GLOSSIFIER, 215  
GRAPPLIFIER, 212  
NULLIFIER, 214  
PREFIXIFIER, 210  
PROMPTIFIER, 220  
SCRUNCHIFIER,  
217-218  
STRETCHIFIER, 219  
PATCHIFIER (AWIIe) patch,  
206

PATCHIFIER (AWIIe)  
program, 8-9, 99-100,  
178-179

.pd8 command, 4, 24, 27, 28,  
34, 65, 83, 149-150

PEEK command, 8

PGLOSS command, 6, 89

Pictures

processing of, 8  
.pi (page interval), 30, 31  
.pl (page length), 30

Platens for printers, 52

POKE command, 8

Pointers, 131, 132, 140

Post processing, 163, 164

Prebooting of ProDOS  
Applewriter 2.0, 32

Prefixes

changing, 20, 138

PREFIXIFIER patch, 103, 210

Print commands

imbedding, 53-55

Printer(s)

accessories for, 41-42  
buffers and, 45  
daisywheel versus dot  
matrix, 49-50

Diablo compatible  
printer, 22

Diablo 630 daisywheel,  
49, 54, 59, 75

Dot matrix printers  
(DMP), 6

Epson printers, 5-6, 54,  
59

getting the computer to  
communicate with a,  
44-45

laser, 39

making adjustments on,  
51

NEC Spinwriters, 49

proportional spacing and,  
49-50

Qume Sprint  
daisywheels, 49

underlining and  
superscripting on,  
5-7

Printing

of a { < , 12  
camera-ready print  
mode, 50-51  
of columns, 15  
of daisywheel spokes,  
20-21

onto a disk, 4

double headers or  
footers, 27

of glossaries, 27-28

of hidden lines on a  
mailing list, 26

imbedding dot  
commands and, 12

improving daisywheel  
print quality, 13

a line of dots, 9

**Printing—cont.**

- microjustification and, 151
- of page numbers, 27
- part of a file, 4, 155-156
- preview mode, 3-4
- problems of Applewriter IIc, 28-29
- ProDOS and, 149-152
- running WPL and, 12
- shadow, 77
- a solid bullet, 34
- underlining and, 33-34, 151
- of WPL files, 27-28

**Print quality**

- accessories and, 41-42
- CAMERA-READY
  - program, 50-51, 64-65, 189
- how to improve, 48-49
- how to improve
  - daisywheel, 13
- importance of, 39-40

**Print tractors, 41, 50****ProDOS Applewriter 2.0**

- backup copies of, 7
- booting of, 114-115
- character entry and, 141-142
- control commands and, 145-149
- detrashing of programs and, 9
- entry points and, 135-136
- erasing programs on
  - boot/master disk, 23
- executing of string
  - control characters, 14
- extended memory and, 23
- filenames and, 10
- grappler problem and, 7
- high work files and, 126-128, 232-233
- HIRES graphics dumps and, 163-164

**ProDOS Applewriter 2.0—**

- cont.
- how to view versions of, 24
- internal files and, 118-119, 128, 234-240
- low work files and, 123-126, 227-231
- memory and, 115, 140
- memory maps and, 116-118
- MLI and, 136-138
- modifying, 161-163
- monitor access and, 138-139
- NULLs and, 5, 96
- patches for, 8, 102-108
- prebooting of, 32
- prefixes and, 20, 103, 138
- printing routines of, 149-152
- reference files and, 128-131, 241-243
- screen display and, 143-145
- storage of work files and, 117
- text files and, 118, 119-123
- underlining and
  - superscripting on, 5-6

**ProDOS Applewriter 2.0**

- programs
- AIOIFIER, 104
- BOOTIFIER, 105
- CONVERT, 11, 112
- CREPIFIER, 106
- CURSIFIER, 108
- GLOSSIFIER, 106
- GRAPPLIFIER, 104-105
- NULLIFIER, 105-106
- PREFIXIFIER, 103
- PROMPTIFIER, 107
- SCRUNCHIFIER, 106-107
- STRETCHIFIER, 107

**ProDOS Applewriter 2.1, 112****PROMPTIFIER patch, 107, 220****Prompts**

- user, 144

**Proportional spacing**

- how to set, 22, 61-62
- microjustification and, 76
- printers and, 49-50

**Punctuation**

- BOLD PS printwheel and, 21
- filenames and, 10
- underlining and, 59-60

**Q****[Q] command, 147****Qume Sprint daisywheels, 49****R****[R] command, 147-148****RAM, 116-118, 140****Random access, 137****READ.EOF commands, 137****Reading a long file, 25****READ.MARK commands, 137****Reference files**

- control commands and, 128-129
- definition of, 119
- error messages, 130
- listing of, 241-243
- function list, 129
- print constants match, 130

**Reformatting, 143****Registration**

- improving daisywheel, 15

**RESTORIFIER (AWIIe)**

- patch, 102, 209

**Ribbons**

- film versus cloth, 50
- how to get more out of, 52-53
- suppliers of, 42-43
- using WD40 on, 52-53

- ROM, 140
- RS232 interface, 44-45, 46-48
- S**
- [S] command, 148
- Saving copy, 10, 148
- Scanning a long file, 25
- Screen display, 143-145
- Screen dump method
  - full, 163, 164
- SCRUNCHIFIER patch, 106-107, 217-218
- Secondline problems of
  - Diablo 630, 22, 75
- Serial interface, 45
- SET.EOF commands, 137
- SET.FILE.INFO., 138
- SET.MARK commands, 137
- SET.PREFIX command, 138
- Shadow printing of titles, 77
- Shortline problem
  - how to solve, 6, 97
- Silencers for printers, 41-42
- SNEAKY.D, 165, 166, 326
- Source code, 32, 164-167
- SPACE ON DISK program, 99
- Spacing
  - in paragraph ends, 76-77
  - tightening vertical, 75-76
  - See also*
    - Microjustification ;
    - Proportional spacing
- Split screen command, 148
- SPOKE REARRANGER
  - program, 21, 75, 182-183
- Squashticity module, 73-74
- STARTUP, 156
- Stashes
  - defined, 131, 132
- STRETCHIFIER (AWIIe)
  - patch, 204
- STRETCHIFIER (AWIIe)
  - program, 6, 61, 73-74, 97-98, 174-176
- STRETCHIFIER (ProDOS 2.0) patch, 107, 219
- String precoding, 163-164
- String variables and WPL, 155
- Subroutines and WPL, 154-155
- Subscripting
  - faking, 14
- Superscripting
  - on Apple DMP, 6
  - on Epson printers, 5-6
  - faking, 14
- Supplies
  - disk labels, 43
  - disk mailers, 43
  - disks, 43
  - paper, 42, 51
  - ribbons, 42-43, 50
- Suppliers
  - list of, 42-43
- Swallow buffer, 31, 125
- .SYS program, 161
- T**
- [T] command, 148
- Tabs
  - [closed apple] key and, 32
  - glossaries and, 61-63
  - high work files and, 126-127
  - how to set, 146, 148
  - low work files and, 126
- TBAS command, 115
- Text files
  - definition of, 118
  - fast typing and, 119-120
  - how to access, 140
  - how to use, 120-123
- TITAN 10, 75
- Titles
  - centering of, 77
- Tractor feeds, 41, 50
- Trashing of programs
  - how to fix, 9
  - punctuation and, 10
- TTXTcommand, 115
- Tutorials
  - creating, 19
- Tutorials—cont.
  - Diablo 630 formatting glossary with, 195-196
  - DMP formatting glossary with, 193-194
  - Epson MX80 formatting glossary with, 197-198
  - how to access, 31
  - Imagewriter formatting glossary with, 199-200
- Tweedle, 156
- TWO COLUMNS, 167-168, 327-328
- Type-ahead buffer, 141
- U**
- [U] command, 148
- Underlining
  - on Epson printers, 5-6
  - formatting and, 71, 74-75
  - how to improve, 7, 33-34, 74-75
  - how to improve
    - daisywheel, 13
    - printing and, 33-34, 151
    - of punctuation, 59-60
- [—] (underscore) command, 149
- US (user separator), 5, 61, 150-151
- V**
- [V] Verbatim command, 14, 55, 56, 59-60, 148, 166-167
- Video cards, 139
- Viewing of a program, 24
- W**
- [W] command, 148
- WD40 for ribbons, 52-53
- Word wraparound command, 149



Word wraparound—cont.  
     defeating, 123  
     STRETCHIFIER patch  
         and, 97

## WPL

    commands and  
         glossaries, 82-83  
     conditional execution  
         and, 155  
     disadvantages of, 157  
     files  
         printing of, 27-28  
     how to write, 154  
     imbedded commands  
         and, 55, 64  
     jumps and subroutines  
         and, 154-155  
     labels and, 153-154  
     numeric variables of, 155  
     printing and, 155-156  
     programs  
         .AUTO.PD8, 78

## WPL—cont.

    .BULLET SHOOTER,  
         34, 186  
     .CAMERA READY, 50-  
         51, 64-65, 78, 188  
     .DETAIL, 78  
     .FILE LISTER, 27-28,  
         184-185  
     .FLINSERT, 35, 187  
     .FORMAT BOLD PS,  
         78  
     .FORMAT MAJESTIC,  
         78  
     .FORMAT NOFRILLS  
         D630, 69-78, 189-191  
     .SPACE ON DISK, 99  
     .SPOKE  
         REARRANGER, 21,  
         75, 182-183

## WPL—cont.

    .TWO COLUMNS,  
         167-168, 327-328  
     string variables of, 155  
     tips on using, 156  
     uses for, 152

## X

[X] command, 148

## Y

[Y] command, 35, 148

## Z

[Z] command, 149

---

# MORE FROM SAMS

---

☐ **Apple® IIc Programmer's Reference Guide**

Describes the four principal programming languages and operating systems: Applesoft BASIC, the monitor, Pro-DOS, and 65C02 machine-language coding. Key topics such as text screen, keyboard input, low- and high-resolution graphics are covered in separate chapters. A complete memory map is included, with procedures for managing all 128K of memory. Valuable for beginners as well as seasoned programmers. David L. Heiserman.  
ISBN 0-672-22422-4.....\$24.95

☐ **Apple® II Plus/IIe Troubleshooting & Repair Guide**

Repair your Apple II or IIe yourself, simply and inexpensively. Troubleshooting flowcharts let you diagnose the probable cause of failure and remedy it. A chapter on advanced troubleshooting shows the more adventuresome how to perform complex repairs. Some knowledge of electronics required. Robert Brenner.  
ISBN 0-672-22353-8.....\$19.95

☐ **Managing with AppleWorks™**

This book makes AppleWorks understandable even for the reader who has no experience with computers or integrated software. Author Ruth K. Witkin provides step-by-step instructions and illustrated examples showing how to use this popular software for effective, efficient business management. Ruth K. Witkin.  
ISBN 0-672-22441-0.....\$17.95

☐ **Applesoft for the IIe**

This book is a detailed Applesoft programmer's reference manual written specifically for the Apple® IIe, covering all aspects of IIe syntax and programming techniques. Contains many usable routines and programs, and offers instant relief from high-priced factory manuals. Blackwood and Blackwood.  
ISBN 0-672-22259-0.....\$19.95

☐ **Apple® IIe Programmer's Reference Guide**

Here's a book that encourages you to explore new programming ideas and take advantage of powerful programming procedures on the Apple IIe by placing needed facts, applications, and other technical information at your fingertips. Also contains many short application and demonstration programs in BASIC and assembly language. David L. Heiserman.  
ISBN 0-672-22299-X.....\$21.95

☐ **Assembly Cookbook for the Apple® II/IIe**

Read one of the strongest, most convincing sermons you'll ever hear in favor of assembly language programming on the Apple. Several chapters of equally strong instruction tell you what assemblers are and how to use them. Step by step, the author leads you through practical modules of working assembly language code. Excellent Lancaster stuff! Don Lancaster.  
ISBN 0-672-22331-7.....\$21.95

---

Look for these Sams Books at your local bookstore.

---

To order direct, call 800-428-SAMS or fill out the form below.

**Please send me the books whose titles and numbers I have listed below.**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
Name (please print) \_\_\_\_\_  
Address \_\_\_\_\_  
City \_\_\_\_\_  
State/Zip \_\_\_\_\_  
Signature \_\_\_\_\_  
(required for credit card purchases)

Enclosed is a check or money order for \$ \_\_\_\_\_  
(plus \$2.00 postage and handling).

Charge my: ☐ VISA ☐ MasterCard

Account No. \_\_\_\_\_ Expiration Date \_\_\_\_\_

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Mail to: Howard W. Sams & Co., Inc.  
Dept. DM  
4300 West 62nd Street  
Indianapolis, IN 46268

DC028

**SAMS™**



## ***The DON LANCASTER Library***

Apple II & IIe Assembly Cookbook .....	22332
Active Filter Cookbook .....	21168
Cheap Video Cookbook .....	21524
CMOS Cookbook .....	21398
Enhancing Your Apple II, Vol I .....	21846
Enhancing Your Apple II, Vol II .....	21943
Hexadecimal Chronicles .....	21802
Incredible Secret Money Machine .....	21562
Micro Cookbook I (Fundamentals) .....	21828
Micro Cookbook II (Machine Language) .....	21829
RTL Cookbook .....	21168
TTL Cookbook .....	21035
Son of Cheap Video .....	21723
TV Typewriter Cookbook .....	21313

### ***Applewriter User Support***

Don Lancaster and Synergetics offer many different support services for Applewriter and Apple users wishing to push the limits of their Apple. Included in these services are feedback cards, companion support disks, free user patches, the Gila Valley Apple Growers Association, and a no-charge voice hotline. An electronic bulletin board and a Compuserve link are also planned.

### ***Feedback Response Cards***

The feedback response cards are your way of letting us know about any problems you may have or telling us what you want to see in the way of future Applewriter enhancements. We also like to hear about your progress in exploring the enhancements. Simply tear out the response card, fill it out, and drop it in the mail.

### ***Companion Disks***

Also available are a pair of companion disks that hold all of the code in this book, along with a few bonus programs.

The 54+ program DOS Version 3.3e disk or the 58+ program ProDOS version costs only \$24.50 or \$44.50 for the pair. Both are unprotected and fully copyable but only for your personal use. You can order these support disks with the order card that is bound in the back of the book. VISA and MasterCard are accepted, as are telephone orders via the helpline.

A number of other books and disk packages are also available, including the support disks for *Enhancing Your Apple II, Volume I*; *Enhancing Your Apple II, Volume II*; *Apple II & IIe Assembly Cookbook*, and *The Incredible Secret Money Machine*, which is Don Lancaster's underground classic book on forming your own winning technical ventures. Also available are some exciting new graphics animation packages. See the order card for full details.

## ***The Gila Valley Apple Growers Association***

Please: Pronounce the G in *Gila* as if it were an H, as in *Hee luh*.

The Gila Valley Apple Growers Association is a most unusual consortium of Apple owners and users. Each meeting is a test ground for the members' software product development. New ways of pushing the limits of Apples and other hardware and software are the usual activity focus. A special interest group on tinaja questing also exists.

Membership is more or less free but is limited strictly and exclusively to those attending the meetings. There are not and never will be any printed notices, newsletters, or outside library exchange services available.

The association meets from 6 to 10 p.m. usually every Wednesday night during the school year. Meetings are held in Thatcher, usually at Eastern Arizona College room T8 or T9.

Stop in sometime.

## ***The Applewriter Helpline***

A voice helpline service is available as a joint service of Don Lancaster, Synergetics, and the Gila Valley Apple Growers Association . . .

### **Applewriter Users Helpline**

---

{602} 428-4073  
(voice only)

The service is free, except for your usual phone charges. Best times to call are weekdays 8 a.m.-5 p.m. (Mountain Standard Time). The two main areas of expertise are Applewriter and assembly language programming.

## ***Cards Missing?***

If the cards are missing, call or write . . .

SYNERGETICS  
746 First Street  
Box 809  
Thatcher, AZ 85552  
{602} 428-4073



## DISKETTE

Please send me:

- ☐ Applewriter IIe Cookbook Disk .....\$24.50
- ☐ ProDOS Applewriter 2.0 Cookbook Disk .....\$24.50
- ☐ Both Applewriter Cookbook Disks .....\$44.50
- ☐ Assembly Cookbook Disk .....\$19.50
- ☐ Enhance Volume I Disk .....\$19.50
- ☐ Enhance Volume II Disk .....\$19.50
- ☐ Absolute Reset Mod package .....\$19.50
- ☐ Vaporlock Instant Sync package .....\$19.50
- ☐ Applewriter Laserwriter Utilities .....\$39.50
- ☐ MacPaint Schematics Toolkit .....\$24.50
- ☐ Old Fangled Animation package .....\$19.50
- ☐ Incredible Secret Money Machine .....\$ 7.50
- ☐ Don Lancaster Book and Software List .....(free)
  
- ☐ Check enclosed for \$\_\_\_\_\_
- ☐ Charge my VISA/MasterCard account \_\_\_\_\_-\_\_\_\_\_-\_\_\_\_\_  
signature \_\_\_\_\_ exp \_\_\_\_\_-\_\_\_\_\_

Name \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

Please: NO purchase orders, COD, cash, Canadian, or foreign.

## RESPONSE CARD

- ☐ Keep me informed of any updates and revisions to the Applewriter Cookbook.
- ☐ Please send me a free Don Lancaster software and book list.
- ☐ The next enhancements I want to see are

---

---

---

---

- ☐ What I need right now is \_\_\_\_\_

---

NAME \_\_\_\_\_

ADDRESS \_\_\_\_\_

CITY \_\_\_\_\_ STATE \_\_\_\_\_ ZIP \_\_\_\_\_

voice phone \_\_\_\_\_ data phone \_\_\_\_\_



\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

place  
postage  
here

**SYNERGETICS**  
**746 First Street**  
**Box 809**  
**Thatcher, AZ, 85552**

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

place  
postage  
here

**SYNERGETICS**  
**746 First Street**  
**Box 809**  
**Thatcher, AZ, 85552**



# AppleWriter™ Cookbook

*AppleWriter Cookbook* gives workable and working answers to real-life questions asked by callers to the Gila Valley Apple Growers Association helpline.

Another in the popular series of Don Lancaster cookbooks, this volume includes the latest ProDOS Version 2.0 details for both the Apple IIc and IIe. It covers a wide range of topics, including:

- Answers to the most-asked AppleWriter questions
- Patches for null, shortline, Grappler®, and other codes
- Self-prompting glossaries for major printers
- Microjustification and proportional spacing routines
- Camera-ready print quality secrets
- Complete and thorough disassembly script
- Source-code capturing instructions
- WPL routines for columns, space-on-disk, etc.
- Information concerning continuing support, helpline, and upgrades

Personalize your copy of AppleWriter so that it does exactly what you want it to do! Anyone who uses AppleWriter needs the *AppleWriter Cookbook*.

**Don Lancaster** heads Synergetics, a prototyping and consulting firm that specializes in microcomputer applications and electronic design. Mr. Lancaster is a prolific author. Two of his best-known books are the classics, *CMOS Cookbook* and *TTL Cookbook*. A microcomputer pioneer, he has a unique writing style that blends information with entertainment so that his readers can smile while they learn.

**Howard W. Sams & Co., Inc.**

A Subsidiary of Macmillan, Inc.

4300 West 62nd Street, Indianapolis, IN 46268 USA



0 81262 22460 1

\$19.95/22460

ISBN: 0-672-22460-7